পাইথন দিয়ে প্রোগ্রামিং শেখা

প্রথম খণ্ড

তামিম শাহ্‌রিয়ার সুবিন

দ্বিমিক প্রকাশনী

পাইথন প্রোগ্রামিং বই

বইয়ের সূচিপত্র

Online Version: http://pybook.subeen.com/

Published by subeen on April 23, 2018

পাইথন প্রোগ্রামিং বই

পাইথন দিয়ে প্রোগ্রামিং শেখার বই। যারা আগে কখনও প্রোগ্রামিং করে নি, তাদের জন্য বইটি উপযোগী। বইটি অনলাইনে ফ্রি পড়া যাবে এবং লিঙ্ক শেয়ার করা যাবে। তবে লেখকের অনুমতি ছাড়া বইয়ের কোনো অংশ নকল করে নিজের ব্লগে বা অন্য কোথাও প্রকাশ করা যাবে না।

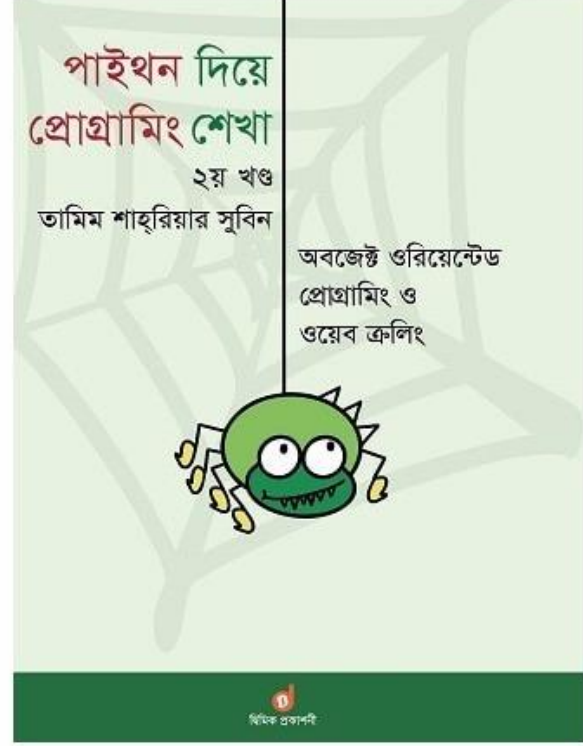
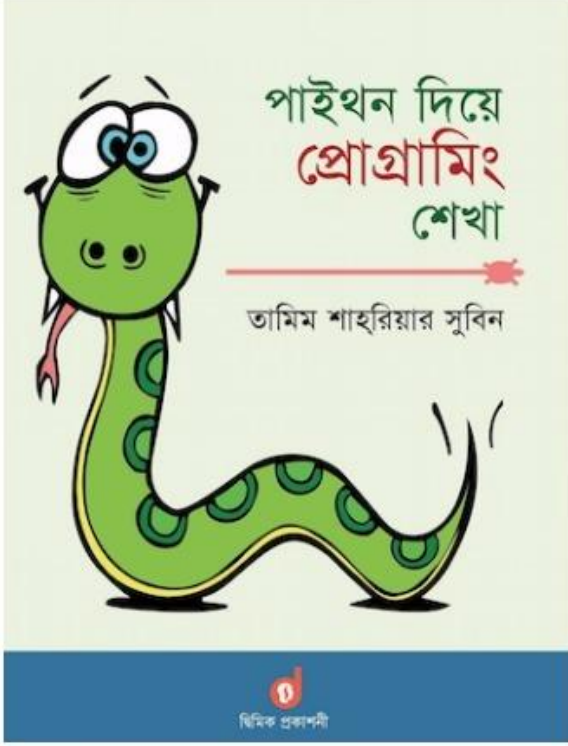
উৎসর্গ

মুহম্মদ জাফর ইকবাল – আমার শিক্ষক ও প্রিয় মানুষ, আমার বই লেখার অনুপ্রেরণা। স্যারের উৎসাহ না থাকলে আমার কখনও লেখালেখি করা হতো না। স্যারের জন্য ভালোবাসা।

ভূমিকা

বর্তমান বিশ্বে পাইথন একটি অত্যন্ত জনপ্রিয় প্রোগ্রামিং ভাষা। বিশ্বের অন্যান্য দেশের মতো বাংলাদেশেও পাইথনের প্রতি আগ্রহ দিনদিন বৃদ্ধি পাচ্ছে। আমি নিজে ২০০৭ সালে পাইথন শেখা শুরু করি। সেসময় মাঝেমধ্যে একটি ব্লগে (Life is short – you need python) পাইথন বিষয়ক লেখালেখি করতাম, যেটি সেই সময়ে বেশ জনপ্রিয় ছিল। পরবর্তি কালে ২০১৩ সালে পাইথন নিয়ে একটি ভিডিও টিউটোরিয়াল সিরিজ তৈরি করি, আর সেই ভিডিওর কনটেন্ট নিয়ে ২০১৫ সালে একটি বইও প্রকাশ করি – পাইথন পরিচিতি নামে। বইটিতে পাইথন ২ ব্যবহার করা হয়েছিল এবং এটি ছিল যারা প্রোগ্রামিং করতে পারে, তাদের জন্য। বইটি মোটামুটি জনপ্রিয়তা অর্জন করে। বর্তমানে পাইথন ২ এর ব্যবহার হ্রাস পাচ্ছে।

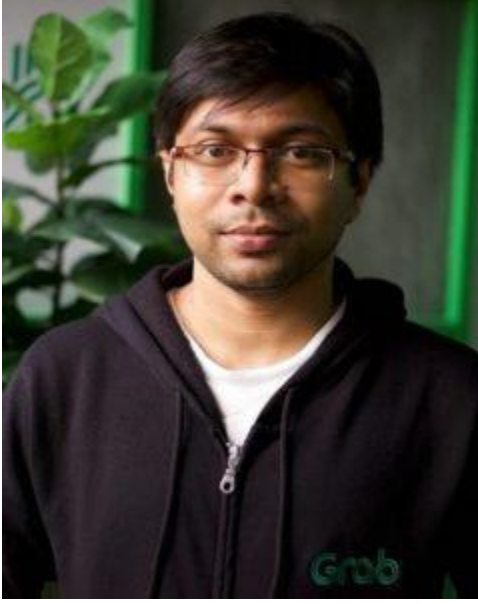
এদিকে যারা নতুন প্রোগ্রামিং শিখবে, তাদের জন্য ২০১১ সালে আমার কম্পিউটার প্রোগ্রামিং ১ম খণ্ড বইটি প্রকাশিত হয় – যেটি সারা পৃথিবীর বাংলাভাষাভাষী শিক্ষার্থীদের কাছে বিপুল জনপ্রিয়তা অর্জন করে। বইটির অনলাইন সংস্করণ ইন্টারনেটে ফ্রি পাওয়া যায় সিপিবুক ওয়েবসাইটে। বইতে সি প্রোগ্রামিং ভাষা ব্যবহার করে শিক্ষার্থীদেরকে প্রোগ্রামিংয়ের সঙ্গে পরিচয় করিয়ে দেওয়া হয়েছে। প্রোগ্রামিং শুরু করার জন্য অনেক বছর থেকেই সি প্রোগ্রামিং ভাষা সারা পৃথিবীজুড়ে ব্যবহার করা হচ্ছে। তবে বেশ কয়েকবছর ধরে অনেক জায়গাতেই প্রথম প্রোগ্রামিং ভাষা হিসেবে পাইথন ব্যবহার করা শুরু হয়েছে – যার ফলে আরো অধিক সংখ্যক শিক্ষার্থী প্রোগ্রামিং শেখায় আগ্রহী হচ্ছে। তাই বাংলাদেশেও যেন শিক্ষার্থীরা পাইথন দিয়ে প্রোগ্রামিং শেখা শুরু করতে পারে,

সেজন্য ২০১৭ সালে দ্বিমিক প্রকাশনী থেকে আমার লেখা পাইথন দিয়ে প্রোগ্রামিং শেখা বইটি প্রকাশিত হয়।

এই বইটিও শিক্ষার্থীদের কাছে অল্প সময়েই জনপ্রিয় হয়। একই বছর আমি এই বইয়ের দ্বিতীয় খণ্ড: পাইথন দিয়ে প্রোগ্রামিং শেখা ২য় খণ্ড – অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ও ওয়েব ক্রলিং বইটি প্রকাশ করি। আবার ২০১৭ সালেই পলিটেকনিক ইনস্টিটিউটগুলোর সিলেবাসে পাইথন নিয়ে আসা হয়, আর সেকারণে তাহমিদ রাফি'র সঙ্গে মিলে প্রোগ্রামিং এসেনশিয়ালস – পাইথন ৩ নামে একটি বই লিখি, তাদের সিলেবাসের সঙ্গে মিল রেখে। ২০১৮ সালের জানুয়ারি মাসে "পাইথন দিয়ে প্রোগ্রামিং শেখা ৩য় খণ্ড – ডেটা স্ট্রাকচার ও অ্যালগরিদম" বইটি প্রকাশিত হবে, যা শিক্ষার্থীদেরকে দক্ষ প্রোগ্রামার হিসেবে গড়ে উঠতে সহায়তা করবে বলে আমার বিশ্বাস।

পাইথন নিয়ে লেখা বইগুলো বাংলাদেশের ভেতরে জনপ্রিয়তা অর্জন করলেও নানান কারণে দেশের বাইরে (মাঝে-মধ্যে দেশের ভেতরেও) শিক্ষার্থীদের কাছে বইগুলো পৌঁছতে পারছে না – যার মূল কারণ হচ্ছে পাইথন এখনও শিক্ষার্থীদের কাছে যথেষ্ট পরিচিত নয়, তার ওপর বাংলাদেশের শিক্ষার্থীরা বই কেনার ব্যাপারে অনিচ্ছুক – তারা নিশ্চিত না হয়ে বই কিনতে চায় না। সেজন্য আমি পাইথন দিয়ে প্রোগ্রামিং শেখা বইয়ের দুটি খণ্ড (প্রথম খণ্ড ও দ্বিতীয় খণ্ডের প্রথম দুটি অধ্যায়) মিলে একটি অনলাইন সংস্করণ তৈরির সিদ্ধান্ত নিই। দ্বিমিক প্রকাশনীর প্রকাশক তাহমিদ রাফি এ ব্যাপারে ইতিবাচক সাড়া দেন। এই অনলাইন সংস্করণের মাধ্যমে শিক্ষার্থীরা সঠিকভাবে পাইথনের সঙ্গে পরিচিত হতে পারবে বলে আমি মনে করি। আশা করি, বইটি লক্ষ লক্ষ বাংলাভাষাভাষী ছেলেমেয়েদের প্রোগ্রামিংয়ের আনন্দময় জগতের সঙ্গে পরিচিত করাবে।

লেখক পরিচিতি

Published by subeen on April 29, 2018

তামিম শাহ্‌রিয়ার (ডাকনাম : সুবিন)-এর জন্ম ১৯৮২ সালের ৭ নভেম্বর ময়মনসিংহে। গ্রামের বাড়ি কুমিল্লা জেলার চান্দিনা উপজেলার হারং গ্রামে। তাঁর বাবা মো: মোজাম্মেল হক ছিলেন সরকারি কর্মকর্তা এবং মা ফেরদৌসি বেগম গৃহিণী। স্ত্রী সিরাজুম মুনিরা ও পুত্র আরাভ শাহরিয়ারকে নিয়ে বর্তমানে সিঙ্গাপুরে বসবাস করছেন।

লেখাপড়া করেছেন হোমনা সরকারি প্রাথমিক বিদ্যালয়, এ কে উচ্চ বিদ্যালয়, নটর ডেম কলেজ এবং শাহজালাল বিজ্ঞান ও প্রযুক্তি বিশ্ববিদ্যালয়ে। ২০০৬ সালে শাহজালাল বিজ্ঞান ও প্রযুক্তি বিশ্ববিদ্যালয়ে কম্পিউটার সায়েন্স ও ইঞ্জিনিয়ারিং বিভাগ থেকে পাস করেছেন।

বিশ্ববিদ্যালয়ে থাকাকালীন বিভিন্ন প্রোগ্রামিং প্রতিযোগিতায় অংশগ্রহণ করেছেন। পরবর্তী সময়ে (২০০৭ ও ২০০৮ সালে) তিনি এসিএম আইসিপিসি ঢাকা রিজিওনাল-এর বিচারক ছিলেন। একটি বেসরকারি বিশ্ববিদ্যালয়ে শিক্ষকতা দিয়ে কর্মজীবন শুরু করলেও পরে সফটওয়্যার প্রকৌশলী হিসেবে কাজ শুরু করেন। বাংলাদেশে থাকাকালীন সময়ে প্রতিষ্ঠা করেছেন মুক্ত সফটওয়্যার লিমিটেড ও দ্বিমিক কম্পিউটিং। এ ছাড়া তিনি বাংলাদেশ গণিত অলিম্পিয়াডে একজন একাডেমিক কাউন্সিলর।

বর্তমানে সিঙ্গাপুরে গ্র্যাব নামক একটি আন্তর্জাতিক প্রতিষ্ঠানে ইঞ্জিনিয়ারিং ম্যানেজার হিসেবে কাজ করছেন।

প্রোগ্রামিং ও পাইথন

Published by subeen on April 22, 2018

প্রোগ্রামিং কী?

কম্পিউটার একটি আশ্চর্য যন্ত্র যা অত্যন্ত দ্রুতগতিতে অনেক হিসেব-নিকেশ করতে পারে। তার ওপর প্রচুর পরিমাণ তথ্যও ধারণ করতে পারে, তার স্থায়ী ও অস্থায়ী মেমোরিতে। তো এই সুবিধাকে কাজে লাগিয়ে বিভিন্ন সমস্যা সমাধানের জন্য মানুষ তৈরি করছে হরেক রকমের সফটওয়্যার। সেই সফটওয়্যার হতে পারে একটি সাধারণ ক্যালকুলেটর, কিংবা চালকবিহীন গাড়ি চালানোর সফটওয়্যার, একটি শিক্ষা প্রতিষ্ঠানের ভর্তি ব্যবস্থা, কিংবা ব্যবসা প্রতিষ্ঠা পরিচালনার জন্য তৈরি সফটওয়্যার। কত কাজে যে মানুষ সফটওয়্যার ব্যবহার করে, সেটি বলে শেষ করা যাবে না। তো সফটওয়্যার ব্যবহারের সুবিধা কী? যেসব কাজে প্রচুর পরিমাণ তথ্য প্রক্রিয়া (process) ও সংরক্ষণ করার কাজ করতে হয়, যা অনেক সময় সাপেক্ষ, সেখানেই সফটওয়্যার ব্যবহার করে অত্যন্ত দ্রুত ও নির্ভুলভাবে কাজটি করা যায়। এসএসসি পরীক্ষার ফলাফল প্রকাশের কথাই ধরা যাক। সেখানে ১০-১৫ লক্ষ শিক্ষার্থীর অনেকগুলো বিষয়ের পরীক্ষার ফলাফল প্রক্রিয়া করতে হয়, তারপর চূড়ান্ত ফল প্রকাশ করতে হয়। সফটওয়্যারের মাধ্যমে কাজটি করার ফলে হাজার হাজার ঘণ্টার শ্রম বেঁচে যায়।

মানুষ যে এত সফটওয়্যার তৈরি করছে, মানুষ নিজে কী করবে? উত্তরে সবাই বলেন যে, মানুষ বিভিন্ন সৃজনশীল কাজ করবে, যদিও কৃত্রিম বুদ্ধিমত্তা (ইংরেজিতে বলে Artificial Intelligence) দিয়ে কম্পিউটারও একটু একটু করে সৃজনশীল হওয়া আরম্ভ করেছে। কিন্তু যত বেশি সফটওয়্যার তৈরি করতে হবে, রক্ষণাবেক্ষণ করতে হবে, তত বেশি প্রোগ্রামারের দরকার হবে – কেবল বাংলাদেশেই নয়, সারা বিশ্বে। বিশ্বব্যাপী ভালো প্রোগ্রামারদের রয়েছে অনেক চাহিদা। তবে একটি বিষয় মনে রাখতে হবে, প্রোগ্রামার হওয়া তেমন কোনো কঠিন কাজ না হলেও ভালো প্রোগ্রামার হওয়া অনেক কঠিন কাজ। এর জন্য বছরের পর বছর ধরে লেখাপড়া ও চর্চা করতে হয়।

প্রোগ্রামাররা কী করে? প্রোগ্রামাররা সফটওয়্যার তৈরি করে, যেখানে এক বা একাধিক প্রোগ্রাম থাকে। প্রোগ্রামগুলোতে থাকে সোর্স কোড, যা কম্পিউটার বুঝতে পারে, এমন কোনো ভাষায় লিখতে হয়। সোর্স কোডে একটি নির্দিষ্ট ভাষায় কম্পিউটারের জন্য কিছু নির্দেশ দেওয়া থাকে, যা আরেকটি সফটওয়্যারের মাধ্যমে কম্পিউটার বুঝে নেয় এবং সেই অনুসারে কাজ করে।

সবাই তো সফটওয়্যার তৈরি করবে না। তাহলে সবার কি প্রোগ্রামিং শেখা উচিত? উত্তর হচ্ছে হ্যাঁ, এখন সবারই প্রোগ্রামিং শেখা উচিত। প্রোগ্রামিং চর্চা মানুষের মস্তিষ্ককে শানিত করে, যুক্তি দিয়ে কোনো সমস্যা বা তথ্য বিশ্লেষণের অভ্যাস তৈরি করে। এছাড়া উচ্চশিক্ষার বেশিরভাগ বিষয়েই নিজে প্রোগ্রামিং করার প্রয়োজন হয়, তাই আগে থেকে প্রোগ্রামিং জানা থাকা উচিত। আর অচিরেই

কর্মক্ষেত্রেও নানান কাজে প্রোগ্রামিং করার প্রয়োজন হবে। তাই স্কুল-কলেজের সকল শিক্ষার্থীরই প্রোগ্রামিং শেখা প্রয়োজন।

পাইথন কী?

বিভিন্ন প্রোগ্রামিং ভাষার মধ্যে পাইথন হচ্ছে অত্যন্ত জনপ্রিয় একটি ভাষা। পাইথন ব্যবহার করে ওয়েব, মোবাইল, ডেস্কটপ ইত্যাদি প্ল্যাটফর্মের জন্য সফটওয়্যার তৈরি করা যায়। মেশিন লার্নিং (Machine Learning) ও ডেটা সায়েন্স (Data Science) ক্ষেত্রেও পাইথন অনেক জনপ্রিয়। আর প্রোগ্রামিং শেখাকে সহজতর করার জন্যও পাইথন ব্যবহার করা হচ্ছে অনেক জায়গায়। পাইথন ভাষা তৈরি করেন গুইডো ভন রুযাম (Guido van Rossum)। ১৯৮৯ সালে পাইথন তৈরির কাজ শুরু হয়, এখনও পাইথন নিয়মিত হালনাগাদ হচ্ছে এবং এর বিভিন্ন সংস্করণ (ভার্সন) প্রকাশিত হচ্ছে। প্রোগ্রামিং শেখার জন্য যেমন পাইথন গুরুত্বপূর্ণ, একাডেমিক কাজে যেমন পাইথন ব্যবহার করা হয়, তেমনি সফটওয়্যার ইন্ডাস্ট্রিতেও পাইথনের অনেক ব্যবহার রয়েছে। এই মুহূর্তে বাংলাদেশে ও বাংলাদেশের বাইরে ভালো পাইথন প্রোগ্রামারের অনেক চাহিদা। তবে লক্ষ করতে হবে, শুধু পাইথন প্রোগ্রামার নয়, বরং ভালো পাইথন প্রোগ্রামার।

এই বইটি কাদের জন্য?

যারা আগে কখনো প্রোগ্রামিং করে নি, পাইথন দিয়ে প্রোগ্রামিং শেখা শুরু করতে চায়, তাদের জন্য বইটি লেখা হয়েছে। অষ্টম শ্রেণী কিংবা তার ওপরের শ্রেণীর কোনো শিক্ষার্থীর বইটি পড়তে সমস্যা হওয়ার কথা নয়। বইটি পড়ার জন্য খুব বেশি গণিতও জানতে হবে না। তবে পাঠকদের প্রতি আমার পরামর্শ হবে, পঞ্চম, ষষ্ঠ, সপ্তম ও অষ্টম শ্রেণীর গণিত বইগুলো আরেকবার পড়ে নেওয়ার জন্য। তাতে এই বইটি পড়তে যেমন সুবিধা হবে, জীবনের অন্যান্য ক্ষেত্রেও এটি কাজে লাগবে। বইটি পড়ার সময় অবশ্যই কম্পিউটার সঙ্গে থাকতে হবে। কম্পিউটারে যেকোনো প্রচলিত অপারেটিং সিস্টেম থাকলেই চলবে, যেমন উইন্ডোজ, ম্যাক ওএসএক্স, লিনাক্স (উবুন্টু, মিন্ট কিংবা অন্য যেকোনো ডিস্ট্রিবিউশন)। তবে আমার পরামর্শ হবে লিনাক্সভিত্তিক কোনো অপারেটিং সিস্টেম ব্যবহার করার জন্য। তাতে প্রোগ্রামিং জীবন অনেক সহজ ও আনন্দময় হয়। আর যাদের কম্পিউটার নেই, তারা অ্যান্ড্রয়েড চালিত মোবাইল ফোনেও প্রোগ্রামিং করতে পারে, QPython সফটওয়্যারের সাহায্যে।

পাইথন ইনস্টল করা

লিনাক্সভিত্তিক বেশিরভাগ অপারেটিং সিস্টেমে পাইথন এমনিতেই ইনস্টল করা থাকে। উইন্ডোজ ব্যবহারকারীদের জন্য পাইথন ডাউনলোড করে ইনস্টল করে নিতে হবে। পাইথন ডাউনলোডের জন্য ওয়েবসাইট
হচ্ছে https://www.python.org/downloads/ (https://www.python.org/downloads/ I) I এখা

নে গিয়ে পাইথন 3.5 (বা তার চেয়ে নতুন যেকোনো সংস্করণ) ডাউনলোড করে ইনস্টল করতে হবে।

পাইথন ছাড়াও কম্পিউটারে একটি টেক্সট এডিটর সফটওয়্যার ইনস্টল করা থাকলে ভালো হয়। নিচে ডাউনলোড করার লিঙ্কসহ দুটি টেক্সট এডিটরের নাম দেওয়া হলো, যেকোনো একটি ইনস্টল করলেই হবে:

- Atom (https://atom.io/\)
- Visual Studio Code (https://code.visualstudio.com/\)

যারা উইন্ডোজ ব্যবহার করে, তারা কিভাবে পাইথন ইনস্টল করবে ও পাইথন দিয়ে প্রোগ্রামিং করবে, সেটি পরিশিষ্ট অংশে দেখানো হয়েছে।

পাইথন দিয়ে প্রথম প্রোগ্রাম

Published by subeen on April 21, 2018

পাইথন দিয়ে প্রথম প্রোগ্রাম

পাইথন ব্যবহার করে প্রথম প্রোগ্রাম লেখার আগে আরো কিছু কথা বলা প্রয়োজন। প্রোগ্রামিং শিক্ষার্থীদের একটি বহুল জিজ্ঞাসিত প্রশ্ন হচ্ছে, আমি তো সফটওয়্যার বানাতে চাই (কারণ শিক্ষার্থীদের ধারণা, সফটওয়্যার বলতে গ্রাফিক্যাল ইউজার ইন্টারফেস সমৃদ্ধ সফটওয়্যার বোঝায়), কিন্তু কমান্ড প্রম্পট (command prompt) বা টার্মিনাল (terminal)-এ প্রোগ্রাম লিখে কী লাভ? আসলে গুই (গ্রাফিক্যাল ইউজার ইন্টারফেস) হচ্ছে প্রোগ্রামিং জগতের খুব ছোট্ট একটি অংশ। এটি শেখার চেয়ে প্রোগ্রামিং ভাষা ও লজিক শেখা ও চর্চা করাই অনেক গুরুত্বপূর্ণ। গান শেখার আগেই যেমন সঙ্গীতশিল্পীরা অ্যালবাম বের করে না, বরং কয়েকবছর গান শিখে, চর্চা করে, তারপরেই ওই কাজে হাত দেয়, সফটওয়্যার তৈরির ব্যাপারটিও তেমনি। আমরা পূর্ণাঙ্গ সফটওয়্যার অবশ্যই বানাবো, তবে সেটি ভালোভাবে প্রোগ্রামিং শেখার পরে।

দ্বিতীয় প্রশ্ন হচ্ছে, প্রোগ্রামিং শেখার সময় সবকিছুতেই প্রশ্ন : "কেন"? (এটি কেন হয়, সেটি কেন হয় – এরকম আর কি)। এক্ষেত্রে আমার পরামর্শ হচ্ছে প্রোগ্রামিং শেখার শুরুর দিকে "কী হয়" সেদিকেই বেশি জোর দেওয়া। কোন কাজ করলে কী হয় এবং বিভিন্ন প্রশ্নের উত্তর খোঁজার জন্য নিজেই প্রোগ্রাম লিখে পরীক্ষা করে দেখা। প্রোগ্রামিং মোটামুটি শেখা হলে, তারপর বিভিন্ন "কেন" প্রশ্নের উত্তর খোঁজা যাবে। শুরুতেই "কেন" নিয়ে অস্থির থাকলে প্রোগ্রামিংটা আর শেখা হবে না।

এখন আমরা আমাদের প্রথম প্রোগ্রাম লিখবো। সারা বিশ্বের সঙ্গে তাল মিলিয়ে আমাদের প্রোগ্রামটিও হবে একটি ছোট্ট প্রোগ্রাম, যা Hello World কথাটি স্ক্রিনে প্রিন্ট করবে। এটি বিশ্বব্যাপী প্রচলিত একটি রীতি আর কি। যাদের কম্পিউটারে উইন্ডোজ অপারেটিং সিস্টেম আছে, তারা পাইথন সফটওয়্যারটি চালু করবে আর যারা লিনাক্স বা ম্যাক ব্যবহারকারী, তারা টার্মিনাল চালু করবে এবং সেখানে python লিখে এন্টার কী চাপবে। তাহলে পাইথন ইন্টারপ্রেটার চালু হবে। সেখানে পাইথন ভাষায় লেখা যেকোনো নির্দেশ বা কমান্ড সঙ্গে সঙ্গে রান করে আউপুট দেখা যাবে। আমরা লিখব – print("Hello World")। এর মাধ্যমে আমরা কম্পিউটারকে বলছি যে স্ক্রিনে Hello World লেখাটি দেখাতে।

```
>>> print("Hello World")
Hello World!
>>>
```

তাহলে আমরা আমাদের প্রথম প্রোগ্রাম লিখে ফেললাম! পাঠককে অভিনন্দন, কারণ প্রোগ্রামিং শেখার পথে একটি গুরুত্বপূর্ণ ধাপ আমরা পার করলাম।

এখন আমরা এই কাজটিই আবার করবো, তবে ভিন্নভাবে। আমরা প্রোগ্রামটি একটি ফাইলে সেভ করবো। তারপরে সেই ফাইলটি পাইথন ব্যবহার করে চালাবো। এজন্য যেকোনো টেক্সট এডিটরে পাইথন কোড লিখে ফাইলে সেভ করতে হবে আর ফাইলের এক্সটেনশন দিতে হবে .py। যেমন আমরা hello.py নামে একটি ফাইল তৈরি করবো, যেখানে কেবল নিচের কমান্ডটি লেখা থাকবে :

```
print("Hello World")
```

তারপরে ফাইলটি সেভ করতে হবে।

এখন আমরা টার্মিনালে গিয়ে python hello.py কমান্ড দিলে প্রোগ্রামটি রান হবে, আর আউটপুট দেখা যাবে : Hello World।

তাহলে আমরা এতক্ষণে শিখে ফেললাম কিভাবে পাইথন ইন্টারপ্রেটার ব্যবহার করে প্রোগ্রাম লিখতে হয়, আবার কিভাবে আলাদা ফাইলে পাইথন প্রোগ্রাম লিখে সেটি চালাতে হয়।

প্রতিটি ভাষার যেমন কিছু নিয়মকানুন রয়েছে, তেমনি প্রোগ্রামিং ভাষারও নিয়মকানুন রয়েছে। আমরা যখন প্রোগ্রাম লিখে রান করতে যাই, তখন যেই ভাষায় প্রোগ্রাম লেখা হয়েছে, তার নিয়মকানুন পরীক্ষা করা হয় এবং কোনো ভুল পাওয়া গেলে প্রোগ্রাম বন্ধ হয়ে সেই ভুলের কথা জানিয়ে দেওয়া হয়। এখন আমরা একটি ভুল করে দেখবো যে ভুল করলে কী হয়। পূর্বে তৈরি করা hello.py ফাইলে আমরা Hello World এর শুরুতে এবং শেষে ডবল কোটেশন চিহ্ন ব্যবহার করেছি। কেন করেছি, সেই আলোচনা পরে করবো, তবে এখন দেখবো যে শুরুতে ডবল কোটেশন চিহ্ন এবং শেষে সিঙ্গেল কোটেশন চিহ্ন ব্যবহার করলে কী হয়? এজন্য আমরা সেই ফাইলে শেষ ডবল কোটশন চিহ্নটি পরিবর্তন করে সিঙ্গেল কোটেশন চিহ্ন ব্যবহার করবো :

```
print("Hello World')
```

এখন ফাইলটি সেভ করে প্রোগ্রাম রান করার চেষ্টা করলে নিচের মতো আউটপুট আসবে :

```
File "hello.py", line 1
print("Hello World')
^
SyntaxError: EOL while scanning string literal
```

পাইথন আমাদের বলছে যে, প্রোগ্রামে সিনট্যাক্স এরর হয়েছে, সেই সঙ্গে কোথায় ভুল হয়েছে, সেটিও দেখিয়ে দিলো। সঙ্গে আবার একটি এরর মেসেজও দিল, যদিও আমরা এখন সেটি ভালোভাবে বুঝতে পারছি না। যাই হোক, এরকম (এবং আরো অনেকরকম) ভুল আমাদের প্রায়ই হবে, তবে তাতে ভয় পেলে চলবে না। ভুলগুলো ঠিকঠাক করে প্রোগ্রাম আবার চালাতে হবে।

ভ্যারিয়েবল, ডেটা টাইপ ও গাণিতিক অপারেশন

Published by subeen on April 20, 2018

আমরা ইতিমধ্যেই জেনে গিয়েছি যে, প্রোগ্রামের কাজ হচ্ছে বিভিন্ন রকমের তথ্য প্রক্রিয়া করা। তো এই তথ্য প্রক্রিয়া করার জন্য সেটি প্রথমে কম্পিউটারের মেমোরিতে রাখতে হবে এবং তারপরে বিভিন্ন অপারেশন চালাতে হবে। যেমন, ধরা যাক, আমাদেরকে দুইটি সংখ্যা দেওয়া হলো এবং আমাদেরকে সেগুলোর যোগফল বের করে দেখাতে হবে। এখন, যেই দুইটি সংখ্যা যোগ করতে হবে, সেগুলোকে তো কোথাও রাখতে হবে, আর যোগফলকেও এক জায়গায় রাখতে হবে। এজন্য আমরা ব্যবহার করবো ভ্যারিয়েবল (variable)। ভ্যারিয়েবলে আমরা বিভিন্ন ধরনের ডেটা রাখতে পারি। আমরা পূর্বের অধ্যায়ে করা প্রোগ্রামটি এখন ভ্যারিয়েবল ব্যবহার করে করবো, এরপর যোগফল বের করার প্রোগ্রামটি লিখবো। আমাদের কোড হবে এরকম :

```
text_to_print = "Hello World"
print(text_to_print)
```

এটি রান করলে আমরা পূর্বের মতো স্ক্রিনে Hello World লেখা দেখতে পাবো। এখানে text_to_print হচ্ছে একটি ভ্যারিয়েবল, যেখানে আমি Hello World কথাটি জমা রাখলাম। তারপরে সেটি প্রিন্ট করলাম। ভ্যারিয়েবলের নাম দেওয়ার কিছু নিয়মকানুন আছে, যেগুলো বিভিন্ন টেক্সট বইতে দেওয়া আছে। আমরা এখন সেইসব নিয়ম মুখস্থ করতে যাবো না, বরং প্রোগ্রাম লিখতে লিখতে আর সেই সঙ্গে ভুল করতে করতে আমরা শিখবো। এখন text_to_print না লিখে আমরা অন্য কোনো নামও ব্যবহার করতে পারতাম, যেমন a, b, c, s ইত্যাদি। কিন্তু আমাদের খেয়াল রাখতে হবে যেন ভ্যারিয়েবলের নাম অর্থপূর্ণ হয়, মানে ভ্যারিয়েবলের নাম দেখে যেন বুঝতে পারা যায় যে, সেটি কী কাজে ব্যবহৃত হচ্ছে।

এখন আমরা টার্মিনালে পাইথন চালু করে আরো কিছু প্রোগ্রাম লিখবো। প্রথম প্রোগ্রাম হচ্ছে দুইটি সংখ্যা যোগ করার প্রোগ্রাম। দুইটি সংখ্যা রাখার জন্য আমরা number1 ও number2 নামে দুইটি ভ্যারিয়েবল ব্যবহার করবো। আর তাদের যোগফল রাখবো result নামে আরেকটি ভ্যারিয়েবলে। তারপরে সেটি প্রিন্ট করবো।

```
>>> number1 = 10
>>> number2 = 5
>>> result = number1 + number2
>>> print(result)
15
```

আমরা উপরে একটি গাণিতিক অপারেশন করলাম, যেটি হচ্ছে যোগ (+)। তেমনি আমরা বিয়োগ (-), গুণ (*) ও ভাগের (/) কাজও করতে পারি। যারা কেবল প্রোগ্রামিং শিখছে, তাদের প্রতি আমার

পরামর্শ হচ্ছে যে, আমি বইতে যেসব কাজ করে দেখাচ্ছি, এগুলো কেবল পড়লে হবে না, সেই সঙ্গে কম্পিউটারে নিজে করে দেখতে হবে। তাহলে প্রোগ্রামিং শেখা সহজ হবে।

আমরা number1 = 10 যে লিখলাম, সেটি বোঝায় যে, number1 নামের ভ্যারিয়েবলে আমরা 10 সংখ্যাটি রাখতে চাচ্ছি। এখন আমরা যদি এরপর number1 ভ্যারিয়েবলে অন্য কিছু রাখতে চাই, সেটিও সম্ভব। নিচের উদাহরণ লক্ষ করলে বিষয়টি পরিষ্কার হবে :

```
>>> number1 = 10
>>> print(number1)
10
>>> number1 = 15
>>> number1 = 18
>>> print(number1)
18
>>> number1 = "hello"
>>> print(number1)
hello
>>>
```

আমরা দেখতে পাচ্ছি যে, একটি ভ্যারিয়েবলে আমরা বিভিন্ন রকম জিনিস রাখতে পারি, তবে সর্বশেষ যেটি রাখবো, প্রোগ্রাম কেবল সেটিই মনে রাখবে। আমরা একটি ভ্যারিয়েবলকে একটি গ্লাসের সঙ্গে তুলনা করতে পারি। একটি গ্লাসে আমি পানি রাখতে পারি। আবার সেই গ্লাসে দুধ রাখতে পারি, শরবত রাখতে পারি কিংবা অন্য কোনো পানীয়। একটি গ্লাস একটি নির্দিষ্ট সময়ে যেকোনো এক প্রকার তরল পদার্থ ধারণ করবে। তেমনি একটি ভ্যারিয়েবল একটি ডেটা ধারণ করবে।

এখন আরেকটি প্রোগ্রাম লেখার চেষ্টা করি।

```
>>> number1 = "hello"
>>> number2 = 2
>>> result = number1 + number2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

এখানে যোগ করার সময় পাইথন বলছে যে, একটি এরর হয়েছে। এররটি কী? এখন দেখে আমাদের খুব বেশি বোঝার কথা নয়। কারণ আমরা কেবল প্রোগ্রামিং শেখা শুরু করেছি।

প্রতিটি ভ্যারিয়েবলেই আমরা যে ডেটা রাখি, তার একটি নির্দিষ্ট ধরন রয়েছে। যেমন 10 হচ্ছে একটি পূর্ণসংখ্যা (ইংরেজিতে integer)। “hello” হচ্ছে একটি স্ট্রিং (string)। তো ভ্যারিয়েবলের টাইপ পাইথন কিভাবে বুঝে? সে ডেটা দেখে একটা ধারণা করে নেয়, আমাদেরকে নিজ থেকে বলে দিতে হয় না। আমরা চাইলে কোনো ভ্যারিয়েবলে সংরক্ষিত ডেটার টাইপ দেখতে পারি।

```
>>> v = 10
>>> type(v)
```

```
<class 'int'>
>>> v = "10"
>>> type(v)
<class 'str'>
>>> v = 10.0
>>> type(v)
<class 'float'>
```

ওপরে আমরা তিন ধরনের ডেটা টাইপ দেখতে পাচ্ছি – int, str ও float। এগুলোর পূর্ণ রূপ হচ্ছে integer, string ও float। integer হচ্ছে যেকোনো পূর্ণসংখ্যা, আর float হচ্ছে যেকোনো বাস্তব সংখ্যা বা দশমিক যুক্ত সংখ্যা। আর string শব্দের অর্থ আমাদের কাছে দড়ি (বা রশি) মনে হলেও এটি আসলে একটি ডেটা টাইপ যেখানে এক বা একাধিক অক্ষর (সংখ্যা, চিহ্ন বা বর্ণ) থাকে। স্ট্রিংয়ে আমাদের হয় সিঙ্গেল কোটেশন চিহ্ন ব্যবহার করতে হবে (যেমন : ‘abc’), অথবা ডবল কোটেশন চিহ্ন ব্যবহার করতে হবে (যেমন : “abc”)। সিঙ্গেল কোটেশন দিয়ে শুরু, কিন্তু ডবল কোটেশন দিয়ে শেষ কিংবা তার উল্টোটি করা যাবে না। আবার খালি স্ট্রিংও সম্ভব (যেমন : “” বা ‘’)। আমরা এখন স্ট্রিংয়ের আরো কিছু উদাহরণ দেখবো।

```
>>> s = "100"
>>> print(s)
100
>>> s = "abcd1234-09232<>?323"
>>> print(s)
abcd1234-09232<>?323
>>> s = 'abc 123'
>>> print(s)
abc 123
>>> s = ''
>>> print(s)

>>> s = ""
>>> print(s)

>>>
```

আমরা যখন কোনো গাণিতিক অপারেশন করি, যেমন যোগ-বিয়োগ-গুণ-ভাগ, তখন যেই দুটি সংখ্যার ওপর গাণিতিক অপারেশন করা হয়, তাদেরকে একই ডেটা টাইপের হতে হয়, আর না হলে পাইথন নিজে চেষ্টা করে তাদেরকে একই ডেটা টাইপে রূপান্তর করে তারপরে গাণিতিক অপারেশন করতে। যেমন, আমরা যদি একটি ইন্টিজার ও একটি ফ্লোট ভ্যারিয়েবল যোগ করি, তাহলে যোগফল হবে ফ্লোট। কিন্তু ইন্টিজার ও স্ট্রিংয়ের যোগের বেলাতে পাইথন স্ট্রিংয়ের সঙ্গে ইন্টিজার যোগ করতে পারবে না।

```
>>> a = 1
>>> b = 2.5
>>> c = a + b
>>> print(c)
3.5
>>>
>>> a = 1
>>> b = "2"
```

```
>>> c = a + b
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

এখন আমরা টার্মিনালে আরো কিছু কোড লিখে কিছু গাণিতিক অপারেটরের ব্যবহার দেখে নেব। এগুলো মুখস্থ করার প্রয়োজন নেই। ঠিকঠাক চর্চা করলে এমনিতেই মনে থাকবে আর মনে না থাকলে বই বা ইন্টারনেট ঘেঁটে দেখে নিলেও হবে।

```
ভাগ করা -
>>> 5 / 2
2.5
বিয়োগ করা -
>>> 5 - 2
3
গুণ করা -
>>> 5 * 2
10
নিচের কাজটিও ভাগ, তবে ভাগফলের কেবল ইন্টিজার অংশ পাওয়া যাবে
>>> 5 // 2
>>> 2
মডুলাস কিংবা ভাগশেষ -
>>> 5 % 2
1
বর্গ করা -
>>> 5 ** 2
25
```

এখন আমাদের আরেকটি বিষয় জেনে নিতে হবে। আমরা যে print() লিখে কোনোকিছু স্ক্রিনে দেখাচ্ছি, এখানে print() হচ্ছে একটি ফাংশন (function)। এই ফাংশনের ভেতরে আমরা যা পাঠাবো, ফাংশনটি তা স্ক্রিনে প্রিন্ট করবে বা দেখাবে। এখন ফাংশনের ভেতরে যা পাঠানো হয়, তাকে বলে ফাংশনের আর্গুমেন্ট (argument)। print() ফাংশনটি কিভাবে কাজ করবে, সেটি কিন্তু আমাদের জানতে হবে না, পাইথনের যে সফটওয়্যার প্রোগ্রামটি চালায়, সেখানে ওই ফাংশনের কাজ বলা আছে। তাই একে আমরা বলতে পারি বিল্ট-ইন (built-in) ফাংশন। print() ফাংশনটি কিন্তু এক বা একাধিক আর্গুমেন্ট নিতে পারে। সেক্ষেত্রে আর্গুমেন্টগুলোকে কমা চিহ্ন দিয়ে পৃথক করে দিতে হবে।

```
>>> a = 1
>>> b = 1.2
>>> c = "hello"
>>> print(a, b, c)
1 1.2 hello
>>> print(a, b, c, "world")
1 1.2 hello world
>>>
```

print() ছাড়াও আমরা ইতিমধ্যে আরেকটা ফাংশন ব্যবহার করেছি, সেটি হচ্ছে type()। type() ফাংশনটি আমাদেরকে কোনো ভেরিয়েবলের ডাটা টাইপ বলে দেয়। এরকম আরো কিছু বিল্ট-ইন ফাংশন রয়েছে, আমরা প্রয়োজনমতো ব্যবহার শিখে নেব।

আমরা যদি ব্যবহারকারির কাছ থেকে কোনো কিছু ইনপুট নিতে চাই, তাহলে তার জন্য পাইথনে input() নামে একটি ফাংশন রয়েছে। আমরা এটি ব্যবহার করে একটি সহজ প্রোগ্রাম লিখবো। এজন্য নিচের কোডটি হুবহু টাইপ করে welcome.py নামক ফাইলে সেভ করতে হবে।

```
name = input("What is your name? ")
print("Welcome to Python,", name, "!")
```

এখন টার্মিনালে python welcome.py লিখে এন্টার কি চাপলে নিচের লাইন প্রিন্ট হবে :

What is your name?

তারপরে নিজের নাম টাইপ করে এন্টার চাপতে হবে : What is your name? Tamim Shahriar Subeen।

তাহলে নিচের মতো আউটপুট আসবে :

Welcome to Python, Tamim Shahriar Subeen !

আমরা এতক্ষণ যা শিখলাম, সেগুলো দিয়ে এখন আরেকটি প্রোগ্রাম লিখবো।

```
number1 = input("Please type an integer and press enter: ")
number2 = input("Please type another integer and press enter: ")

print("number1 + number2 =", number1+number2)
```

ওপরের কোড আমরা arithmetic.py নামক ফাইলে সেভ করে রান করবো : python arithmetic.py কমান্ড দিয়ে।

```
Please type an integer and press enter: 3
Please type another integer and press enter: 6
number1 + number2 = 36
```

কিন্তু ফলাফল তো আসার কথা 9। তাহলে এখানে কী হচ্ছে? input() ফাংশনটি সবসময় স্ট্রিং রিটার্ন করে। তাই number1 ও number2 এখানে স্ট্রিং। আর দুটি স্ট্রিংকে আমরা যদি যোগ করি, তাহলে সেগুলো পাশাপাশি বসে আরেকটি নতুন স্ট্রিং তৈরি করবে। তাই ভ্যারিয়েবলগুলোকে ইন্টিজারে রূপান্তর করতে হবে। এজন্য আমরা int() ফাংশন ব্যবহার করতে পারি। এটিও একটি বিল্টইন ফাংশন।

```
number1 = input("Please type an integer and press enter: ")
number2 = input("Please type another integer and press enter: ")
number1 = int(number1)
number2 = int(number2)
print("number1 + number2 =", number1+number2)
```

এখন প্রোগ্রাম রান করলে ঠিকঠাক আউটপুট আসবে।

অনুশীলনী:

- উপরের প্রোগ্রামটি পরিবর্তন করতে হবে যেন সকল প্রকারের গাণিতিক অপারেশন দেখানো যায় (+, -, *, /, %, //, **)।
- এই অধ্যায় পড়ার সময় নিশ্চয়ই মনে অনেক প্রশ্ন জেগেছে। সেগুলোর উত্তর নিজে নিজে খুঁজে বের করার চেষ্টা করতে হবে (বিভিন্ন প্রোগ্রাম লিখে)।

কন্ডিশনাল লজিক

Published by subeen on April 19, 2018

এই অধ্যায়ে আমরা পরিচিত হবো কন্ডিশনাল লজিকের সঙ্গে। এখানে কন্ডিশন হচ্ছে শর্ত। আর কন্ডিশনাল লজিকে এক বা একাধিক শর্ত থাকতে পারে। এক বা একাধিক শর্ত নিয়ে যেই শর্তটি গঠন করা হবে, তার ফলাফল হবে দুই রকম – সত্য (true) বা মিথ্যা (false)। কিছু উদাহরণ দেওয়া যাক।

ধরা যাক, আমি ধানমণ্ডি থেকে রিকশা চড়ে নিউমার্কেট যাবো। আমি রিকশাচালককে ভাড়া জিজ্ঞাসা করায় তিনি বললেন, ভাড়া ৪০ টাকা, একদাম। এখন আমি রিকশায় ওঠার আগে একটি শর্ত পূরণ করতে হবে। সেটি হচ্ছে, আমার কাছে কি কমপক্ষে ৪০ টাকা আছে? এর উত্তর দুই রকম হতে পারে – হ্যাঁ (সত্য) কিংবা না (মিথ্যা)। যদি সত্য হয়, তাহলে আমি রিকশায় উঠবো, মিথ্যা হলে উঠবো না।

আবার ধরা যাক, আমার কয়েকটি বই কেনা প্রয়োজন এবং এর জন্য আমি নীলক্ষেতে (ঢাকার যে এলাকায় অনেক বইয়ের দোকান আছে) গিয়ে বই কেনার সিদ্ধান্ত নিয়েছি। এখন মঙ্গলবার নীলক্ষেতের মার্কেট বন্ধ থাকে। তাই আমি একটি শর্ত দিয়ে সিদ্ধান্ত নেবো যে, আজকে বই কিনতে যাবো কী যাবো না। শর্তটি হচ্ছে, "আজকে মঙ্গলবার"। এটি যদি সত্য হয়, তাহলে আমি যাবো না, মিথ্যা হলে যাবো। আবার ধরা যাক, নীলক্ষেতের দোকানগুলো খোলা থাকে সকাল ১০টা থেকে রাত ৮টা পর্যন্ত। তাহলে আমাকে অবশ্যই আরেকটি শর্ত পূরণ করতে হবে। "আমি রাত ৮টার মধ্যে নীলক্ষেতে পৌঁছতে পারবো" – এই শর্ত সত্য হলে আমি যাবো, মিথ্যা হলে আমি যাবো না। এখন আমরা দেখতে পাচ্ছি এখানে মোট দুইটি শর্ত আছে –

- আজকে মঙ্গলবার
- আমি রাত ৮টার মধ্যে নীলক্ষেতে পৌঁছতে পারবোতাহলে আমি যেকোনোদিন বই কেনার কথা চিন্তা করলে চার ধরনের সম্ভাব্য ঘটনা আমাকে চিন্তা করতে হবে :

১) আজকে মঙ্গলবার নয় এবং আমি রাত ৮টার মধ্যো নীলক্ষেতে পৌঁছাতে পারবো না

২) আজকে মঙ্গলবার নয় এবং আমি রাত ৮টার মধ্যো নীলক্ষেতে পৌঁছাতে পারবো

৩) আজকে মঙ্গলবার এবং আমি রাত ৮টার মধ্যো নীলক্ষেতে পৌঁছাতে পারবো না

৪) আজকে মঙ্গলবার এবং আমি রাত ৮টার মধ্যো নীলক্ষেতে পৌঁছাতে পারবো

ওপরের ঘটনাগুলোর মধ্যে কেবল দ্বিতীয় ঘটনাতেই আমি বই কিনতে নীলক্ষেত যাওয়ার সিদ্ধান্ত নেব। অর্থাৎ "আজকে মঙ্গলবার" শর্তটি মিথ্যা ও "আমি রাত ৮টার মধ্যে নীলক্ষেতে পৌঁছতে পারবো" শর্তটি সত্য হলেই কেবল আমি যাবো।

এখন একটি জ্যামিতির উদাহরণে আসা যাক। আমরা জানি, একটি চতুর্ভূজের সবগুলো কোণ প্রতিটি যদি এক সমকোণ হয়, তাহলে সেটি আয়তক্ষেত্র বা বর্গক্ষেত্র হবে। যদি সবগুলো বাহু সমান হয়, তাহলে বর্গক্ষেত্র আর যদি বিপরীত বাহুগুলো সমান হয় (যেহেতু প্রতিটি কোণ এক সমকোণ, তাই সবগুলো বাহু সমান না হলে বিপরীত বাহুগুলো অবশ্যই সমান হবে), তাহলে আয়তক্ষেত্র।

তাহলে একটি চতুর্ভূজ বর্গক্ষেত্র কী না, সেটি নির্ভর করে দুইটি শর্তের ওপর :

- সবগুলো কোণ প্রতিটি এক সমকোণ (90 ডিগ্রী)
- সবগুলো বাহু সমান দৈর্ঘ্যেরওপরের দুইটি শর্তই সত্য হলে চতুর্ভূজটি হবে একটি বর্গক্ষেত্র, প্রথমটি সত্য ও দ্বিতীয়টি মিথ্যা হলে চতুর্ভূজটি হবে আয়তক্ষেত্র।

এখন আমরা ট্রাফিক সিগন্যালের একটি উদাহরণ দিই। সিগন্যালে লাল, হলুদ কিংবা সবুজ বাতি জ্বলে। লাল বা হলুদ বাতি জ্বললে গাড়ি থামবে, সবুজ বাতি জ্বললে গাড়ি চলবে। তাহলে এখানে শর্তটি কেমন হবে? শর্তটি হতে পারে "সবুজ বাতি জ্বালানো আছে"। এটি সত্য কিংবা মিথ্যা হতে পারে। সত্য হলে গাড়ি চলবে, মিথ্যা হলে গাড়ি থামবে। আবার শর্তটি চাইলে আমরা দুইটি শর্ত ব্যবহার করেও লিখতে পারতাম। "লাল বাতি জ্বালানো আছে", "হলুদ বাতি জ্বালানো আছে" – এই দুইটি শর্তের যেকোনো একটি সত্য হলেই আমরা গাড়ি থামাবো, আর দুইটি শর্তই যদি মিথ্যা হয়, তাহলে গাড়ি চলবে।

এখন কম্পিউটার প্রোগ্রামিংয়ে শর্ত বা কন্ডিশনাল লজিক নিয়ে কাজ করতে হলে কিছু মৌলিক বিষয় আমাদের জানতে হবে। প্রথমটি ইতিমধ্যেই আমরা জেনে গিয়েছি যে এক বা একাধিক শর্ত মিলে আমরা যেই কন্ডিশনাল লজিক তৈরি করবো, তার চূড়ান্ত ফলাফল দুই রকমের হতে পারে: সত্য অথবা মিথ্যা।

দ্বিতীয় বিষয় হচ্ছে, একটি শর্তের উল্টো শর্ত তৈরি করার পদ্ধতি, যাকে বলে NOT (নট)। যেমন "সবুজ বাতি জ্বালানো আছে" শর্তটি আমরা উল্টে ফেলতে পারি এভাবে : "সবুজ বাতি জ্বালানো নেই", এবং একে এভাবেও লেখা যায় : NOT সবুজ বাতি জ্বালানো আছে। "সবুজ বাতি জ্বালানো আছে" যদি সত্য হয়, তাহলে "NOT সবুজ বাতি জ্বালানো আছে" হবে মিথ্যা। আর "সবুজ বাতি জ্বালানো আছে" যদি মিথ্যা হয়, তাহলে "NOT সবুজ বাতি জ্বালানো আছে" হবে সত্য। তেমনি "আজকে মঙ্গলবার" শর্তটি সত্য হলে "NOT আজকে মঙ্গলবার" হবে মিথ্যা। আর "আজকে মঙ্গলবার" শর্তটি মিথ্যা হলে "NOT আজকে মঙ্গলবার" হবে সত্য। তাহলে আমরা বলতে পারি : NOT true হচ্ছে false আর NOT false হচ্ছে true।

আমরা যখন একাধিক শর্ত নিয়ে কাজ করবো, তখন দুইটিই সত্য বা দুইটির যেকোনো একটি সত্য, এমন চিন্তাভাবনা করার প্রয়োজন হয়। তখন আমাদের কাজে লাগবে AND (অ্যান্ড) এবং OR (অর)। AND-এর ক্ষেত্রে যদি তার দুইপাশের (বাম ও ডানপাশের) শর্ত সত্য হয়, তাহলে পুরো শর্তটি

সত্য, এর অন্যথা হলে (যেকোনো একটি মিথ্যা কিংবা দুটিই মিথ্যা) পুরো শর্তটি মিথ্যা। যেমন : কোনো চতুর্ভূজ বর্গক্ষেত্র হওয়ার শর্ত হচ্ছে : সবগুলো কোণ প্রতিটি এক সমকোণ (90 ডিগ্রী) AND সবগুলো বাহু সমান দৈর্ঘ্যের। অর্থাৎ, AND এর বামদিকের ও ডানদিকের দুইটি শর্তই সত্য হতে হবে।

OR-এর ক্ষেত্রে, তার দুইপাশের যেকোনো একটি সত্য হলেই পুরো শর্তটি সত্য। কেবল দুইটিই যদি মিথ্যা হয়, তাহলে শর্তটি মিথ্যা। যেমন: ট্রাফিক লাইটের ক্ষেত্রে, লাল বাতি জ্বলছে OR হলুদ বাতি জ্বলছে – এই শর্তটি সত্য হবে যদি লাল বাতি জ্বালানো থাকে, কিংবা হলুদ বাতি জ্বালানো থাকে, কিংবা দুটি বাতিই জ্বালানো থাকে। লাল ও হলুদ দুটি বাতিই যদি জ্বালানো না থাকে, তাহলে শর্তটি মিথ্যা।

ওপরের আলোচনার সারসংক্ষেপ আমরা তৈরি করতে পারি নিচের টেবিল ব্যবহার করে।

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

এখন আমরা দেখবো, পাইথনে বিভিন্ন কন্ডিশনাল লজিকের ব্যবহার। দুইটি জিনিস সমান কী না, তা পরীক্ষা করার জন্য আমরা == অপারেটর ব্যবহার করতে পারি। ছোট-বড় পরীক্ষার জন্য আছে >, <, >=, <=। আবার অসমান বোঝানের জন্য আছে != চিহ্ন। কোনটা কী কাজ করে, সহজে বোঝার জন্য আমরা কোড লিখে দেখতে পারি :

```
>>> 2 == 3
False
>>> 3 == 3
True
>>> 2 > 3
False
>>> 2 < 3
True
>>> 2 != 3
True
>>> 3 != 3
False
>>> 2 >= 3
False
>>> 2 <= 3
True
>>>
```

ওপরে আমি print() ফাংশন ব্যবহার করি নি, কারণ আমি যখন পাইথন ইন্টারপ্রেটার চালু করবো, সেখানে কোনো স্টেটমেন্ট চালালে তার ফলাফল পরের লাইনে আপনাআপনি দেখায়। আর আমি যেমন সরাসরি বিভিন্ন সংখ্যার তুলনা করেছি, সেগুলো ভ্যারিয়েবলে রেখে ভ্যারিয়েবলগুলোর মধ্যেও এই তুলনা করার কাজটি করা যেত।

আবার দুটি স্ট্রিং সমান কী না, সেটিও আমরা পরীক্ষা করতে পারি :

```
>>> "Bangladesh" == "Bangladesh"
True
>>> "Bangladesh" == "bangladesh"
False
```

এখন আমরা পরিচিত হবো লিস্ট (list)-এর সঙ্গে। আমরা যখন একাধিক ডেটা একসঙ্গে রাখতে চাই, তখন আমরা লিস্ট ব্যবহার করতে পারি। যেমন 1 থেকে 10 পর্যন্ত সংখ্যাগুলো একসঙ্গে রাখতে চাইলে আমরা একটি লিস্টে রাখবো এভাবে : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]। লিস্ট শুরু ও শেষ করতে হয় স্কয়ার ব্র্যাকেট (অনেক সময় একে থার্ড ব্র্যাকেটও বলা হয়) দিয়ে। আর স্কয়ার ব্র্যাকেটের মধ্যে প্রতিটি উপাদান রাখা হয় এবং সেগুলো কমা চিহ্ন (,) দিয়ে পৃথক করা হয়।

```
>>> number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(number_list)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

আমরা চাইলে লিস্টের প্রতিটি উপাদান আলাদাভাবে পেতে পারি। এজন্য আমাদেরকে লিস্টের নাম লিখে তারপর স্কয়ার ব্র্যাকেটের ভেতরে ওই উপাদানের ক্রমিক সংখ্যা লিখতে হবে। এই ক্রমিক সংখ্যাকে প্রোগ্রামিংয়ের ভাষায় বলে ইনডেক্স (index) এবং এটি শুরু হয় 0 থেকে (কোনো কোনো প্রোগ্রামিং ভাষায় 1 থেকেও শুরু হয়)। এখন তাহলে কিছু উদাহরণ দেখে নিই :

```
>>> number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(number_list)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
>>> print(number_list[0])
1
>>> print(number_list[1])
2
>>> print(number_list[9])
10
>>> print(number_list[10])
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

একটি বিষয় খেয়াল করতে হবে যে, আমাদের লিস্টে যদি দশটি উপাদান থাকে, তাহলে ইনডেক্স থাকবে 0 থেকে 9। তাই আমরা যখন number_list[10] প্রিন্ট করার চেষ্টা করেছি, তখন একটি এরর দিয়ে আমাদেরকে জানিয়ে দেওয়া হয়েছে যে লিস্টের ইনডেক্স রেঞ্জের বাইরে। একটি লিস্টে কতটি

উপাদান আছে, সেটি জানার জন্য আমরা len() ফাংশন ব্যবহার করতে পারি। নিচে আমি সার্কভুক্ত দেশগুলোর একটি তালিকা তৈরি করলাম এবং সেখান থেকেও এও বের করলাম যে মোট কয়টি দেশ সার্কের অন্তর্গত :

```
>>> saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal", "India",
"Pakistan", "Sri Lanka"]
>>> print(saarc)
['Bangladesh', 'Afghanistan', 'Bhutan', 'Nepal', 'India', 'Pakistan', 'Sri
Lanka']
>>> print(saarc[0])
Bangladesh
>>> print("Number of countries in SAARC:", len(saarc))
Number of countries in SAARC: 7
>>>
```

এখন কোনো দেশ saarc এর সদস্য, এটি সত্য না মিথ্যা আমরা সহজেই যাচাই করতে পারবো।

```
>>> "Bangladesh" in saarc
True
>>> "China" in saarc
False
>>> "China" not in saarc
True
>>> "India" not in saarc
False
>>>
```

তাহলে আমরা দেখলাম, কোনো উপাদান একটি লিস্টে আছে কী নেই, সেটি খুব সহজেই যাচাই করা যায়। ওপরের উদাহরণ না বুঝলে ঠান্ডা মাথায় মনোযোগ দিয়ে বোঝার চেষ্টা করতে হবে, এটি না বুঝে সামনে আগানো ঠিক হবে না।

if স্টেটমেন্ট

আমরা তো শিখলাম কিভাবে বিভিন্ন শর্ত প্রয়োগ করতে হয়, কিভাবে একাধিক শর্ত একসঙ্গে প্রয়োগ করতে হয়। এখন কোড লেখার পালা। আমরা যদি চাই যে, কোনো শর্ত সত্য হলে একটি কাজ হবে, তাহলে আমাদেরকে if ব্যবহার করতে হবে। if ব্যবহার করার নিয়ম বোঝার জন্য আমরা নিচের কোড লিখবো এবং saarc.py নামক ফাইলে সেভ করবো।

```
saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal", "India", "Pakistan",
"Sri Lanka"]

country = input("Enter the name of the country: ")
if country in saarc:
    print(country, "is a member of SAARC")

print("Program terminated")
```

তাহলে আমরা দেখতে পাচ্ছি, if এর পরে শর্ত লিখতে হয়, তারপরে একটি কোলন চিহ্ন ব্যবহার করতে হয়। আর সেই শর্ত সত্য হলে কী কাজ করা হবে, সেটি লিখতে হয় if এর নিচে একটি tab (কীবোর্ডের বাম দিকে বাটনটি পাওয়া যাবে) দিয়ে। এই কাজটাকে বলে ইনডেনটেশন করা। ঠিকঠাক ইনডেনটেশন করা না হলে পাইথন বুঝতে পারে না যে কী কাজ করতে হবে। ওপরের প্রোগ্রামে if-এর পরে শর্ত সত্য হলে print(country, “is a member of SAARC”) স্টেটমেন্টটি কাজ করবে। আর পরের প্রিন্টের সঙ্গে if-এর কোনো সম্পর্ক নেই। এটি if ব্লকের বাইরে। এখন আমরা প্রোগ্রাম রান করে দেখি :

```
$ python saarc.py
Enter the name of the country: Bangladesh
Bangladesh is a member of SAARC
Program terminated
$ python saarc.py
Enter the name of the country: Japan
Program terminated
```

Bangladesh-এর ক্ষেত্রে প্রিন্ট করলো যে Bangladesh সার্কের সদস্য কিন্তু Japan-এর বেলায় সেটি করল না। কারণ শর্ত মিথ্যা। আর উভয় ক্ষেত্রেই Program terminated প্রিন্ট করলো, কারণ এর সঙ্গে if-এর সম্পর্ক নেই, এটি if ব্লকের বাইরে।

আমাদের কোডে if-এর ভেতরে ইনডেনটেশনের কাজটি না করলে কী হতো? তখন আমরা এরকম এরর পেতাম: IndentationError: expected an indented block। এই পরীক্ষাটি করার কাজ আমি পাঠকের ওপর ছেড়ে দিলাম।

এখন, আমরা Japan-এর বেলায় তেমন কিছু প্রিন্ট করলাম না। কিন্তু আমরা যদি বলে দিতাম যে Japan সার্কের সদস্য নয়, তাহলে ব্যাপারটি একটু ভালো হতো। এজন্য আমরা ব্যবহার করতে পারি else। if-এর শর্ত মিথ্যা হলে else ব্লকের ভেতরে যা করতে বলা হবে, প্রোগ্রাম তা-ই করবে।

```
saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal", "India", "Pakistan",
"Sri Lanka"]

country = input("Enter the name of the country: ")
if country in saarc:
    print(country, "is a member of SAARC")
else:
    print(country, "is not a member of SAARC")

print("Program terminated")
```

এখন আমরা প্রোগ্রাম রান করলে এরকম আউটপুট পাবো :

```
$ python saarc.py
Enter the name of the country: Japan
Japan is not a member of SAARC
Program terminated
```

কোনো প্রোগ্রামে যদি এরকম থাকে যে একের পর এক বিভিন্ন রকম শর্ত পরীক্ষা করতে হবে, তারজন্য আমরা প্রথম if-এর পরে বাকী শর্তগুলো লেখার জন্য elif ব্যবহার করবো। আমরা এখন একটি প্রোগ্রাম লিখবো যেটি পরীক্ষায় প্রাপ্ত নম্বর থেকে গ্রেড বের করবে, আর সেখানে elif এর ব্যবহার দেখবো।

```
marks = input("Please enter your marks: ")
marks = int(marks)

if marks >= 80:
    grade = "A+"
elif marks >= 70:
    grade = "A"
elif marks >= 60:
    grade = "A-"
elif marks >= 50:
    grade = "B"
else:
    grade = "F"

print("Your grade is", grade)
```

এখন প্রোগ্রামটি grade_calculator.py নামক ফাইলে সেভ করে রান করতে হবে। প্রোগ্রামটি ভালোভাবে লক্ষ করলেই if, elif ও else-এর ব্যবহার বুঝতে পারা যাবে। যখনই একটি শর্ত সত্য হবে, তখন কিন্তু আর বাকী elif বা else-এর শর্ত পরীক্ষা হবে না।

অনুশীলনী:

- ব্যবহারকারীর কাছ থেকে একটি সংখ্যা ইনপুট নিতে হবে, আর সেটি ধনাত্মক না ঋণাত্মক, তা বের করতে হবে।
- আমরা জানি যেসব সংখ্যা দুই দিয়ে নিঃশেষে বিভাজ্য, তাদেরকে বলে জোড় সংখ্যা, আর তা না হলে বিজোড় সংখ্যা। একটি সংখ্যা জোড় না বিজোড়, তা বের করার প্রোগ্রাম লিখতে হবে। এরজন্য মডুলাস অপারেটর (%) ব্যবহার করতে হবে।

লিপ ইয়ার (Leap Year)

এখন আমরা কোনো বছর লিপ ইয়ার (বাংলায় বলে অধিবর্ষ) কি না সেটি বের করার একটি প্রোগ্রাম লিখবো। কোনো বছর লিপ ইয়ার হলে সেই বছরে ফেব্রুয়ারি মাস হয় 29 দিনে, আর লিপ ইয়ার না হলে হয় 28 দিনে। কোনো সাল 4 দিয়ে নিঃশেষে বিভাজ্য হলে সেটি লিপ ইয়ার হয়, তবে একটি ব্যতিক্রম আছে। সেই সাল যদি আবার 100 দিয়ে নিঃশেষ বিভাজ্য হয়, তাহলে তাকে 400 দিয়েও নিঃশেষে বিভাজ্য হতে হবে, নইলে সেটি লিপ ইয়ার হবে না। তাহলে আমরা লজিক সাজাতে পারি এভাবে :

```
if year % 4 != 0:
    print("No")
```

```
else:
    if year % 100 == 0:
        if year % 400 == 0:
            print("Yes")
        else:
            print("No")
    else:
        print("Yes")
```

আবার আমরা চাইলে নিচের মতো করেও লজিক সাজাতে পারি।

```
if year % 400 == 0:
    print("Yes")
elif year % 100 == 0:
    print("No")
elif year % 4 == 0:
    print("Yes")
else:
    print("No")
```

আমরা এখানে একটু বুদ্ধি খাটিয়েছি। কারণ year যদি 400 দিয়ে নিঃশেষে বিভাজ্য হয়, তাহলে সেটি অবশ্যই লিপ ইয়ার। আর তা না হলে এটি যদি 100 দিয়ে নিঃশেষে বিভাজ্য হয়, তবে এটি লিপ ইয়ার নয়। কারণ এটি 100 দিয়ে নিঃশেষ বিভাজ্য কিন্তু 400 দিয়ে নয় (400 দিয়ে নিঃশেষে বিভাজ্য যদি হতো তাহলে তো প্রথম if ব্লকেই ঢুকে যেত, এখানে আসতো না)। আর যদি 100 দিয়ে বিভাজ্য না হয়, তখন আমরা পরীক্ষা করছি যে year 4 দিয়ে নিঃশেষে বিভাজ্য কী না। যদি হয়, তাহলে এটি অবশ্যই লিপ ইয়ার (কারণ এটি 100 দিয়ে বিভাজ্য নয়, সেটি ইতিমধ্যেই পরীক্ষিত)। আর যদি 4 দিয়ে নিঃশেষে বিভাজ্য না হয়, তাহলে তো আর ওই বছর লিপ ইয়ার নয়।

এখন এই লিপইয়ারের প্রোগ্রাম আমরা আরেকভাবে করবো।

```
if year % 100 != 0 and year % 4 == 0:
    print("Yes")
elif year % 100 == 0 and year % 400 == 0:
    print("Yes")
else:
    print("No")
```

আমরা এখানে একাধিক শর্ত একসঙ্গে প্রয়োগ করছি and ব্যবহার করে। প্রোগ্রামটির বিস্তারিত ব্যাখ্যায় আর গেলাম না। বুঝে নেওয়ার দায়িত্ব পাঠকের ওপরই ছেড়ে দিলাম। আশা করি, একটু চিন্তা করলে এবং এখন পর্যন্ত বই ঠিকমতো পড়লে প্রোগ্রামটি বুঝতে সমস্যা হবে না।

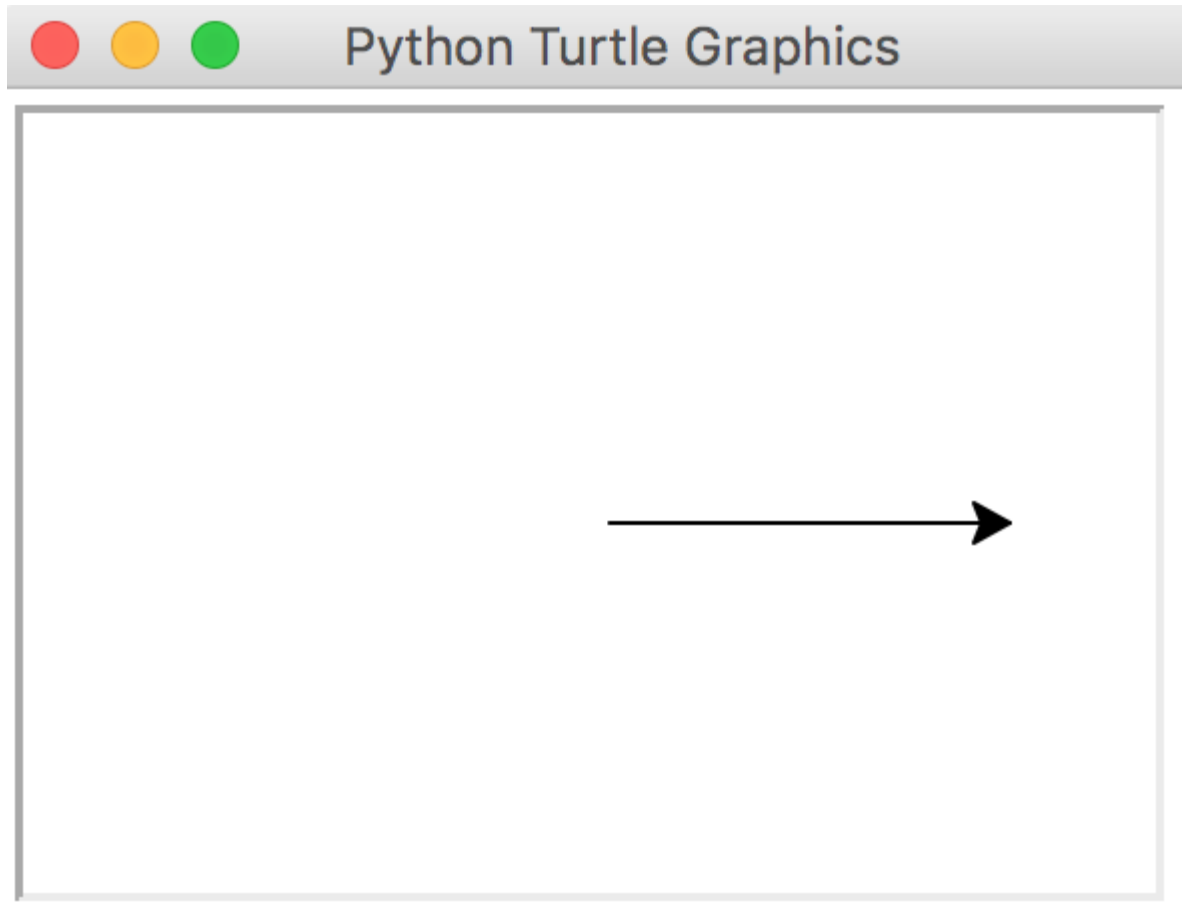
অনুশীলনী: উপরে দেখানো তিনটি প্রোগ্রামই এমনভাবে লিখতে হবে যেন ব্যবহারকারীর কাছ থেকে year ইনপুট নেওয়া হয় এবং লিপ ইয়ার হলে আউটপুটে ওই সালের কথা উল্লেখ করে সেটি লিপ ইয়ার কী না, তা প্রিন্ট করতে হবে।

টার্টলের সঙ্গে পরিচয়

Published by subeen on April 18, 2018

এখন আমরা পরিচিত হবো একটি মজার জিনিসের সঙ্গে। তার নাম হচ্ছে turtle। পাইথনে এই নামে একটি মডিউল আছে, যেটি পাইথনের সঙ্গে ইনস্টল করা থাকে। কারো কম্পিউটার turtle ইনস্টল করা না থাকলে সেটি করে নিতে হবে (পরিশিষ্ট অংশে turtle ইনস্টল করার পদ্ধতি দেওয়া আছে)। আমরা নিচের প্রোগ্রাম লিখে পরীক্ষা করে দেখতে পারি যে, এই মডিউল ইনস্টল করা আছে কী নেই। আর turtle শব্দের অর্থ কচ্ছপ, কিন্তু বইতে আমরা বাংলায় টার্টল নামেই চিনবো, কারণ এটি পাইথনের একটি মডিউল।

```
import turtle

turtle.forward(100)

turtle.exitonclick()
```

যদি এরকম এরর দেয় যে, ImportError: No module named 'turtle' তাহলে বুঝতে হবে যে, এটি ইনস্টল করা নেই। আর যদি কোনো এরর না দেয়, তাহলে নিচের মতো ছবি আসবে :

এখন আসা যাক, আমরা তিনটি লাইনে কী লিখলাম সেই আলোচনায়। প্রথম লাইনে লিখেছি import turtle। এর মাধ্যমে আমরা টার্টল নিয়ে কাজ করার জন্য যেই মডিউল রয়েছে সেটি আমাদের কোডে নিয়ে আসছি। পাইথনে বিভিন্ন কাজ করার জন্য হাজার হাজার মডিউল রয়েছে। আর সেসব মডিউলে বিভিন্ন কাজ ইতিমধ্যে করে দেওয়া আছে। আমরা চাইলে সেসব মডিউল ব্যবহার করতে পারি, কিন্তু তার জন্য আমাদের সেটি import করতে হবে। দ্বিতীয় লাইনে লিখেছি turtle.forward(100)। এই কমান্ড বা স্টেটমেন্ট দিয়ে আমরা টার্টলকে 100 পিক্সেল সামনে যেতে বলছি। সে কীভাবে যাবে, সেটি বলা আছে টার্টল মডিউলের ভেতরে, আমাদের তাই সেটি নিয়ে চিন্তা করতে হবে না। আর হ্যাঁ, পিক্সেল মানে স্ক্রিনের ক্ষুদ্রতম একক, যাকে আমরা বোঝার সুবিধার্থে একটি বিন্দু ধরে নিতে পারি। তৃতীয় লাইনে আমি লিখেছি turtle.exitonclick() যা দিয়ে আমি নির্দেশ দিচ্ছি যে, যেই আউটপুট উইন্ডোটি আসবে, সেখানে যেকোনো জায়গায় ক্লিক করলে প্রোগ্রাম বন্ধ হয়ে যাবে।

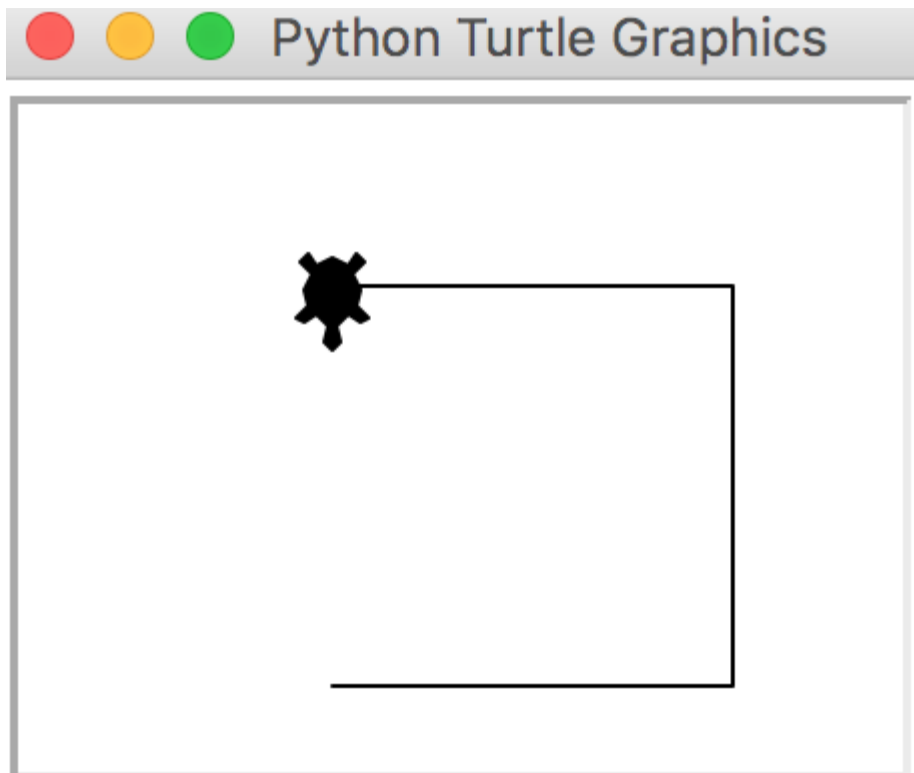
এবারে আমরা দেখবো, টার্টল দিয়ে কিভাবে আমরা একটি বর্গক্ষেত্র তৈরি করতে পারি। প্রতি বাহুর দৈর্ঘ্য যদি 100 পিক্সেল হয়, তাহলে নিচের ধাপগুলো অনুসরণ করতে হবে :

1. প্রথমে 100 পিক্সেল সামনে যেতে হবে।
2. 90 ডিগ্রী বামে ঘুরতে হবে।

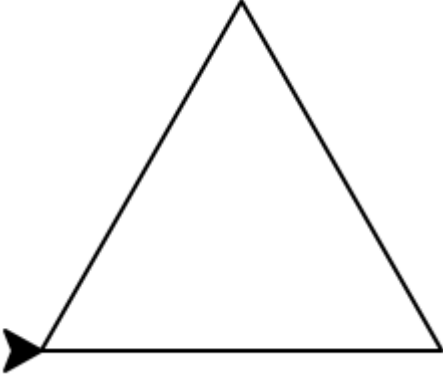
3. 100 পিক্সেল সামনে যেতে হবে।
4. 90 ডিগ্রী বামে ঘুরতে হবে।
5. 100 পিক্সেল সামনে যেতে হবে।
6. 90 ডিগ্রী বামে ঘুরতে হবে।
7. 100 পিক্সেল সামনে যেতে হবে।
8. 90 ডিগ্রী বামে ঘুরতে হবে।

তাহলে আমরা এবার কোড লিখে দেখি।

```
import turtle

turtle.shape("turtle")

turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)

turtle.exitonclick()
```

প্রোগ্রামটি সেভ করে রান করলে নিচের মতো আউটপুট আসবে –

এখানে কিন্তু বর্গক্ষেত্র সম্পূর্ণ হলো না। আরো দুই লাইন (কমপক্ষে এক লাইন) কোড লিখতে হবে। সেই কাজটি আমি পাঠকের জন্য রেখে দিলাম। আরেকটি ব্যাপার খেয়াল করতে হবে যে, এই প্রোগ্রামে কিন্তু টার্টল দেখতে টার্টলের মতোই দেখাচ্ছে। কারণ হচ্ছে প্রোগ্রামের দ্বিতীয় লাইনে সেটি করতে বলা হয়েছে।

আমরা চাইলে টার্টলের গতি নিয়ন্ত্রণ করতে পারি, এর জন্য আমাদেরকে speed() ফাংশন ব্যবহার করতে হবে। এর ভেতরে আর্গুমেন্ট হিসেবে 0 থেকে 10 পাঠানো যাবে। 1 হচ্ছে সবচেয়ে ধীরগতির। যদি এর মান বাড়াই, তাহলে গতি বাড়তে থাকবে, তার মানে 10 অনেক বেশি দ্রুতগতির। আর 0 হচ্ছে সবচেয়ে দ্রুতগতির। আমরা যদি কোনো আর্গুমেন্ট না পাঠাই, তাহলে গতির ডিফল্ট মান 0 হয়, এবং তখন সবচেয়ে টার্টলের গতি হয় সবচেয়ে দ্রুত। বাকীগুলো নিজে নিজে পরীক্ষা করে বের করতে হবে। আগের প্রোগ্রামে turtle.shape(“turtle”) লাইনের পরের লাইনে এভাবে লিখতে হবে : turtle.speed(2)।

অনুশীলনী ঃ টার্টল ব্যবহার করে সমবাহু ত্রিভূজ তৈরি করতে হবে, অর্থাৎ যেই ত্রিভূজের তিনটি বাহুর দৈর্ঘ্যই সমান। আর ছোটদের পাঠ্যবইতে কিন্তু বলা আছে যে সমবাহু ত্রিভূজের ভেতরে প্রতিটি কোণ হবে 60 ডিগ্রী। আউটপুট আসবে নিচের ছবির মতো –

ওপরের প্রোগ্রামটি নিজে করতে না পারলে বুঝতে হবে যে গণিতের মৌলিক জ্ঞানে অনেক ঘাটতি রয়েছে এবং তখন স্কুলের গণিত বই নিয়ে বসে ভালোভাবে পড়তে হবে। গণিতের বেসিক দুর্বল বলে কান্নাকাটি করা চলবে না, বুঝে বুঝে বই পড়লেই বেসিক ভালো হয়ে যাবে।

লুপ – একই কাজ অনেকবার

Published by subeen on April 17, 2018

আমাদের অনেক সময়ই একই কাজ একাধিকবার করতে হয়। যত বেশি বার করতে হয়, আমাদের তত বেশি পরিশ্রম হয়, তত বেশি সময় লাগে। তাই আমরা একই কাজ বারবার করতে পছন্দ করি না। এখন আমরা যদি কম্পিউটারে প্রোগ্রাম লিখে কম্পিউটার দিয়ে কোনো কাজ করাতে চাই, তাহলে তো আমাদের বিভিন্ন স্টেটমেন্ট বা কমান্ড দেওয়া দরকার হয়, তাই না? যেমন কোনো কিছু স্ক্রিনে দেখাতে চাইলে print() ফাংশন লিখতে হয়। এখন আমরা যদি দশবার কোনো জিনিস স্ক্রিনে দেখাতে চাই, তাহলে আমাকে কী দশবার print() ফাংশন দিতে হবে? আপাতত আমাদের উত্তর হচ্ছে হ্যাঁ। এখন, যদি একশ কিংবা এক হাজার বার সেই কাজ করতে চাই, তখন?

প্রোগ্রামিংয়ে একটি জিনিস আছে যার নাম লুপ (loop)। এই লুপের সাহায্যে আমরা একই কাজ বারবার করতে পারি। যেমন আমি যদি দশবার স্ক্রিনে দেখাতে চাই যে I want to be a great programmer, তখন আমরা এভাবে প্রোগ্রাম লিখবো :

```
for i in range(10):
    print("I want to be a great programmer.")
```

এখানে আমি range() এর ভেতরে যত সংখ্যা লিখবো, for এর ভেতরের কাজটি ততবার হবে। ওপরের প্রোগ্রামটি রান করলেই সেটি বুঝতে পারা যাবে। এখন কেউ যদি বলে, I love Bangladesh কথাটি 100 বার প্রিন্ট করতে হবে, তাহলে কীভাবে প্রোগ্রাম লিখতে হবে সেটি নিশ্চয়ই বলে দিতে হবে না?

আমি যখন এভাবে for এর ভেতরে i in range(n) লিখবো, তখন for-এর ভেতরের যেই ব্লক, সেখানে যেসব কাজ করতে বলা হবে, কম্পিউটার সেটি n সংখ্যকবার করবে। এখানে i ভ্যারিয়েবলটি ব্যবহার করার কারণ হচ্ছে i-এর মান 0 থেকে শুরু করে এক এক করে বাড়বে এবং সর্বোচ্চ মান হবে n-1। একটি প্রোগ্রাম লিখে সেটি যাচাই করে নিই :

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

এখন আমরা আগের অধ্যায়ে করা বর্গক্ষেত্র আঁকার প্রোগ্রামে ফেরত যাই। যদিও সেই প্রোগ্রামটি অসম্পূর্ণ ছিল, সেটি ঠিকভাবে করতে পারলে প্রোগ্রামটি হবে এমন :

```
import turtle

turtle.shape("turtle")
turtle.speed(1)

turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)

turtle.exitonclick()
```

এখন আমরা প্রোগ্রামের কোড একটু মনোযোগ দিয়ে দেখলে বুঝতে পারবো যে, এখানে নিচের দুইটি কাজ চারবার করা হচ্ছে:

`turtle.forward(100)`

`turtle.left(90)`

তাহলে আমরা যদি এমনভাবে টার্টলকে নির্দেশ দিই যে, নিচের কাজগুলো চারবার করবে:

- 100 ঘর সামনে যাও।
- 90 ডিগ্রী বামে ঘুর।তাহলেও কিন্তু বর্গক্ষেত্র তৈরি হয়ে যাওয়ার কথা। আমরা সেটি প্রোগ্রাম লিখে দেখি।

```
import turtle

turtle.shape("turtle")
turtle.speed(1)

for i in range(4):
    turtle.forward(100)
    turtle.left(90)

turtle.exitonclick()
```

ওপরের প্রোগ্রামটি চালালে আমরা একটি বর্গক্ষেত্র দেখতে পাবো। এখন কোন কাজগুলো লুপের ভেতরে করতে হবে, সেটি পাইথন বুঝবে কীভাবে? লুপের ভেতরের সব কাজ লুপ যেখানে শুরু হয়েছে, তার এক ট্যাব পরে দিতে হবে। if ব্লকের ভেতরে আমরা যেরকম করেছিলাম আর কী। এই বিষয়টি বুঝতে সমস্যা হলে পরিচিত কারো কাছ থেকে বুঝে নেওয়াই ভালো। তারপরও বইয়ের পরিশিষ্ট অংশে ইনডেনটেশন অধ্যায়ে বিষয়টি আমি আরেকটু বোঝানোর চেষ্টা করেছি।

এখন আমরা একটি ছোট্ট প্রোগ্রাম লিখবো, যার কাজ হবে, পঞ্চাশটি 1 যোগ করা। যোগফল কত হবে? আমরা চট্ করে বলে দিতে পারি যে, পঞ্চাশটি 1-এর যোগফল হবে 50। এখন প্রোগ্রাম লিখে সেটি করবো। আমাদের কিন্তু পঞ্চাশবার যোগ করার স্টেটমেন্ট লিখতে হবে না, কারণ ইতিমধ্যে আমরা লুপ শিখে ফেলেছি। যোগফল রাখার জন্য আমি result নামে একটি ভ্যারিয়েবল ব্যবহার করবো আর লুপের জন্য i। আমরা চাইলে ভ্যারিয়েবলের অন্য নামও দিতে পারি।

```
>>> result = 0
>>> for i in range(50):
...     result = result + 1
...
>>> print(result)
50
>>>
```

result = result + 1 দেখে একটু অদ্ভুত লাগতে পারে। আসলে এখানে প্রথমে ডানপক্ষের বা = চিহ্নের ডান দিকের কাজ আগে হবে, মানে result ভেরিয়েবলের সঙ্গে 1 যোগ হবে, তারপরে সেই মানটি আবার result ভেরিয়েবলেই রাখা হবে। এখন সবার কাছে একটি প্রশ্ন। প্রথমে result = 0 কেন লেখা হলো? উত্তর চিন্তা করে খুঁজে বের করতে হবে। না পারলে পরিচিতজনের সাহায্য নিতে হবে।

এবারে আরেকটু কঠিন একটি সমস্যার সমাধান করার চেষ্টা করবো। সেটি হচ্ছে 1 + 2 + 3 + ... + 48 + 49 + 50-এর যোগফল বের করা। অর্থাৎ 1 থেকে 50 পর্যন্ত প্রতিটি পূর্ণসংখ্যা যোগ করে যোগফল দেখাতে হবে। আগের প্রোগ্রামটির সঙ্গে মিল হচ্ছে যে, এখানেও 50 বার যোগ করতে হবে, তবে পার্থক্য হচ্ছে আগেরবার যেমন প্রতিবার 1 যোগ করেছিলাম, এবারে প্রথমে 1, তারপর 2, তারপর 3 এভাবে 50 পর্যন্ত সংখ্যাগুলো যোগ করতে হবে। এজন্য আমরা result = result + 1 না লিখে লিখবো result = result + num। আর এই num-এর মান প্রথমে হবে 1 তারপরে 2, এভাবে প্রতিবার 1 বাড়বে। প্রোগ্রামটি লিখে ফেলি :

```
>>> result = 0
>>> num = 1
>>> for i in range(50):
...     result = result + num
...     num = num + 1
...
>>> print(result)
1275
```

আমরা যদি আরো খানিকটা চিন্তাভাবনা করি, তাহলে কিন্তু নিচের কোড লিখেই একই কাজ করে ফেলতে পারবো :

```
>>> result = 0
>>> for num in range(51):
...     result = result + num
...
>>> print(result)
```

```
1275
```

যেহেতু আমরা range(51) লিখেছি, তাই num-এর মান 0 থেকে শুরু হবে এবং 50-এ গিয়ে শেষ হবে। এখন একটি জিনিস জেনে রাখা ভালো যে, result = result + num-কে result += num ও লিখা যায়। তেমনি num = num + 1-কে লিখা যায় num += 1।

আমরা দেখলাম যে i in range(n) লিখলে i-এর মান 0 থেকে শুরু হয়। আমরা চাইলে সেটি অন্য কোনো সংখ্যা থেকেও শুরু করতে পারি, এজন্য আমাকে লিখতে হবে এরকম : range(start, n)। তাহলে i-এর মান start থেকে শুরু হয়ে 1 করে বাড়বে এবং n-1-এ গিয়ে শেষ হবে। আমরা যদি একটু চিন্তা করি, আগের প্রোগ্রামে প্রথমবার result = result + num স্টেটমেন্টে আসলে যা হয়েছিল, তা হচ্ছে : result = 0 + 0 (কারণ result-এর মান 0, num-এর মানও 0)। এই কাজটি আমাদের না করলেও চলে। আমরা এখন ওপরে নতুন যা কিছু শিখলাম, তাতে প্রোগ্রামটি নিচের মতো করে লিখতে পারি :

```
>>> result = 0
>>> for num in range(1, 51):
...     result += num
...
>>> print(result)
1275
```

আমরা যদি চাই যে, for লুপের ভেতর কোনো কিছু 1 করে না বেড়ে 5 করে বাড়বে, সেই ব্যবস্থাও আছে। নিচের উদাহরণটি দেখলেই বুঝতে পারবো :

```
>>> for i in range(1, 20, 5):
...     print(i)
...
1
6
11
16
>>>
```

একটি কথা আমি আবারো মনে করিয়ে দিতে চাই যে, এই বইটি পড়ে প্রোগ্রামিং শিখতে চাইলে প্রতিটি উদাহরণ পড়ার সময় নিজে নিজে করে দেখতে হবে। এবং অনুশীলনীগুলোও সব করতে হবে (কোনো সমস্যায় পড়লে বন্ধুর সাহায্য নিতে হবে কিংবা http://programabad.com ওয়েবসাইটে প্রশ্ন করতে হবে)।

এখন আমরা আরেকটি উদাহরণ দেখবো। 1 থেকে 100 পর্যন্ত যেসকল সংখ্যা 5 দ্বারা নিঃশেষে বিভাজ্য (অর্থাৎ 5 দিয়ে ভাগ করলে ভাগশেষ 0 হয়), সেই সংখ্যাগুলো প্রিন্ট করতে হবে এবং তাদের যোগফলও বের করতে হবে।

```
result = 0
for num in range(100):
```

```
    if num % 5 == 0:
        print(num)
        result += num
print("Sum is:", result)
```

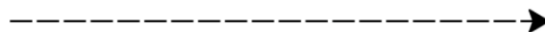
প্রোগ্রামটি আমরা একটি ফাইলে সেভ করে রান করতে পারি (আবার পাইথন ইন্টারপ্রেটারেও রান করতে পারি)। result–এর মান প্রিন্ট হবে 950। কিন্তু সঠিক উত্তর হচ্ছে 1050। এর কারণ কী? আমরা যে range(100) লিখেছি, তাতে তো num-এর মান লুপের ভেতরে 0 থেকে 99 পর্যন্ত হবে। তাহলে আমাদেরকে লিখতে হবে range(101)।

প্রোগ্রামটি আমরা আরো সহজে করতে পারতাম এভাবে :

```
result = 0
for num in range(5, 101, 5):
    print(num)
    result += num
print("Sum is:", result)
```

এখন লুপ ব্যবহার করে আমরা টার্টল দিয়ে একটি ড্যাশ লাইন আঁকবো। এখানে আমরা penup() ও pendown() ফাংশন ব্যবহার করবো। penup() মানে আমি কলম তুলে ফেললাম, এখন টার্টল যা-ই করুক না কেনো, স্ক্রিনে কোনো কিছু আঁকা হবে না। আবার pendown() করলে এর পর থেকে টার্টলের কাজকর্ম স্ক্রিনে দেখাবে। আমরা যেটি করবো, সেটি হচ্ছে 10 পিক্সেল সামনে যাবো, তারপরে penup() করবো, 3 পিক্সেল সামনে যাবো (কিন্তু যেহেতু penup করা হয়েছে, তাই স্ক্রিনে কিছু দেখাবে না), তারপর আবার pendown() করে ফেলবো। এই কাজটা আমাদের একাধিকবার করতে হবে, যত বড় লাইন আঁকতে চাই, তত বেশি বার করতে হবে। নিচের প্রোগ্রামে আমি 20 বার করলাম।

```
import turtle

turtle.speed(1)

for i in range(20):
    turtle.forward(10)
    turtle.penup()
    turtle.forward(3)
    turtle.pendown()

turtle.exitonclick()
```

আউটপুট হবে নিচের মতো :

লুপের ভেতর লুপ

আমাদের প্রয়োজন হলে আমরা লুপের ভেতরও লুপ চালাতে পারি। একে বলে নেস্টেড লুপ (nested loop)। যেমন, আমরা ইতিমধ্যে দেখেছি যে, কিভাবে for লুপ ব্যবহার করে বর্গক্ষেত্র আঁকতে হয়। এখন আমাদের যদি পাঁচটি বর্গক্ষেত্র আঁকার প্রয়োজন পরে, তাহলে কি আমরা বর্গক্ষেত্র আঁকার কোড পাঁচবার লিখবো? অবশ্যই না। আমরা একটি লুপ ব্যবহার করবো যা পাঁচবার ঘুরবে এবং তার ভেতরে বর্গক্ষেত্র আঁকার কোড লিখবো। নিচের কোড দেখে প্রথমে বোঝার চেষ্টা করতে হবে যে, কোডটির আউটপুট কেমন হবে এবং তারপর প্রোগ্রাম লিখে রান করে আউটপুট মিলিয়ে দেখতে হবে।

```
import turtle

turtle.shape("turtle")
turtle.speed(1)

for side_length in range(50, 100, 10):
    for i in range(4):
        turtle.forward(side_length)
        turtle.left(90)

turtle.exitonclick()
```

লিস্টের ওপর লুপ চালানো

আমাদের যদি কোনো লিস্ট থাকে। আমরা সেই লিস্টের ওপর কিন্তু লুপ চালাতে পারি। যেমন :

```
>>> saarc = ["Bangladesh", "Afghanistan", "Bhutan", "Nepal", "India",
"Pakistan", "Sri Lanka"]
>>> for country in saarc:
...     print(country, "is a member of SAARC")
...
Bangladesh is a member of SAARC
Afghanistan is a member of SAARC
Bhutan is a member of SAARC
Nepal is a member of SAARC
India is a member of SAARC
Pakistan is a member of SAARC
Sri Lanka is a member of SAARC
>>>
```

আরেকটি মজার বিষয় হচ্ছে, আমরা range() দিয়েও লিস্ট তৈরি করতে পারি। এজন্য আমাকে list() ফাংশন ব্যবহার করতে হবে। এটিও বিল্ট-ইন ফাংশন, তাই এটি ব্যবহার করার জন্য কোনো মডিউল ইমপোর্ট করতে হবে না। যেমন, আমি যদি চাই, 0 থেকে 10 পর্যন্ত সবগুলো সংখ্যার লিস্ট তৈরি করবো, তাহলে এভাবে করতে পারি :

```
>>> li = list(range(11))
```

```
>>> print(li)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

আবার আমি যদি চাই যে 1 থেকে 20 পর্যন্ত জোড় সংখ্যাগুলোর লিস্ট তৈরি করবো, তাহলে এভাবে করতে পারি :

```
>>> li = list(range(2, 21, 2))
>>> print(li)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

টিকা: এই বইতে আমি list()-কে ফাংশন বলে পরিচিত করালেও ভবিষ্যতে আমরা বিষয়টি আরো জানবো, যখন অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং শিখবো। এই বইতে আমি সেটি নিয়ে আলোচনা করি নি।

while লুপ

এখন আমরা আরেক ধরনের লুপ দেখবো, যার নাম while লুপ। যারা নতুন নতুন প্রোগ্রামিং শিখে, তাদের মনে প্রায়ই প্রশ্ন জাগে, লুপ তো একভাবে লিখলেই হয়, আমার কি সবরকম পদ্ধতি শেখার দরকার আছে? এক কথায় উত্তর হচ্ছে, অবশ্যই দরকার আছে।

while লুপের ক্ষেত্রে প্রথমে while লিখতে হবে, তারপরে একটি শর্ত লিখতে হবে, যেই শর্ত সত্যি হলে লুপের ভেতরের কাজ একবার হবে, তারপরে আবার সেই শর্ত পরীক্ষা করা হবে। এভাবে যতক্ষণ শর্তটি সত্যি হবে, ততক্ষণ লুপের ভেতরের কাজ চলতে থাকবে এবং প্রতিবার লুপের ভেতরের কাজ চলার পরে সেই শর্তও পরীক্ষা করা হতে থাকবে। আমরা এখন দুইটি উদাহরণ দেখবো, তাহলে while লুপ কিভাবে ব্যবহার করতে হয়, সেটি আমাদের কাছে পরিষ্কার হয়ে যাবে।

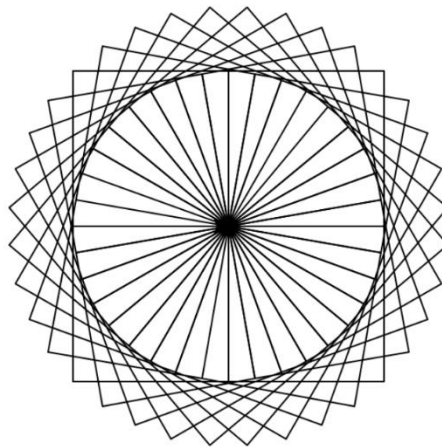
```
>>> i = 0
>>> while i < 5:
...     print(i)
...     i += 1
...
0
1
2
3
4
>>>
>>> i = 5
>>> while i >= 0:
...     i -= 1
...     print(i)
...
4
3
2
1
```

```
0
-1
>>>
```

আমরা এখন while লুপ ব্যবহার করে নামতা প্রিন্ট করার একটি প্রোগ্রাম লিখবো।

```
n = input("Please enter a positive integer: ")
n = int(n)

m = 1
while m <= 10:
    print(n, "x", m, "=", n*m)
    m += 1
```

ওপরের প্রোগ্রামটি namta.py নামে সেভ করে রান করে দেখতে হবে। আমরা চাইলে multiplication_table.py নামেও ফাইলটি সেভ করতে পারতাম, কারণ নামতাকে ইংরেজিতে বলে multiplication table।

আমরা ইতিপূর্বে দেখেছি যে, for লুপের ভেতরে for লুপ ব্যবহার করা যায়। তেমনি for লুপের ভেতরে while লুপ, while লুপের ভেতরে for লুপ, while লুপের ভেতরে while লুপ, এরকম ব্যবহার করা যায়। পরবর্তী প্রোগ্রামে আমরা সেটি করবো। সেই সঙ্গে টার্টল দিয়ে লাইনগুলোর রং পরিবর্তন করা যায়, সেটিও দেখানো হবে। প্রোগ্রামটির আউটপুট হবে এমন :

```
import turtle

turtle.color("black")
turtle.speed(5)

counter = 0
while counter < 36:
    for i in range(4):
        turtle.forward(100)
        turtle.right(90)
    turtle.right(10)
    counter += 1
```

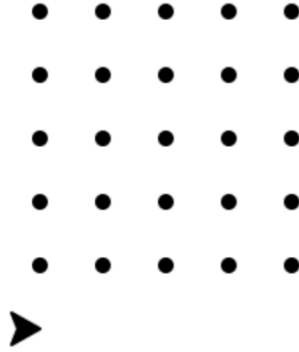
```
turtle.exitonclick()
```

turtle.color(“black”) লাইনে বিভিন্ন রং (যেমন : blue, green, red, yellow, orange ইত্যাদি) ব্যবহার করে প্রোগ্রামটি বারবার চালিয়ে দেখতে হবে। আমি বই ছাপানোর খরচ কমানোর জন্য black ব্যবহার করেছি। এখন সবার কাছে একটি প্রশ্ন। ওপরের প্রোগ্রামে আমি কেন counter < 36 শর্ত ব্যবহার করেছি?

টার্টল ব্যবহার করে লুপের ভেতরে লুপ ব্যবহার করার আরেকটি উদাহরণ দেখে নিই। আমরা যত বেশি প্রোগ্রামিং করব, প্রোগ্রামিং শেখাটা তত ভালো হবে।

```
import turtle

height = 5
width = 5

turtle.speed(2)

turtle.penup()

for y in range(height):
    for x in range(width):
        turtle.dot()
        turtle.forward(20)
    turtle.backward(20 * width)
    turtle.right(90)
    turtle.forward(20)
    turtle.left(90)

turtle.exitonclick()
```

এখানে dot() ফাংশন ব্যবহার করে ডট প্রিন্ট করা হচ্ছে। আর শুরুতেই আমরা penup() কল করেছি যেন forward(), backword() এসব ফাংশনের কাজ হওয়ার সময় স্ক্রিনে কোনো কিছু আঁকা না হয়। মানে কলম একটু ওপরে তুলে যদি কাগজের এদিক-সেদিক নেই, তাহলে তো কাগজে কোনো দাগ পড়বে না, তাই নয় কি? প্রোগ্রামটিতে height, width -এর বিভিন্ন মান দিয়ে এবং x, y এর মান লুপের ভেতরে প্রিন্ট করে পরীক্ষা-নিরীক্ষা করা যেতে পারে। আর প্রোগ্রামের আউটপুট

হবে এমন :

break এবং continue

লুপের ভেতরে কখনও কখনও আমাদের বিশেষ অবস্থার সৃষ্টি হতে পারে, যেই অবস্থার সৃষ্টি হলে আমরা চাই যে, প্রোগ্রাম লুপ থেকে বের হয়ে যাবে। সেই কাজটি করার জন্য আমরা break ব্যবহার করতে পারি। আবার আমরা যদি চাই যে, কোনো বিশেষ অবস্থায় লুপের ভেতরে পুরো কাজ না হয়ে লুপটি চলতে থাকবে, তখন আমরা continue ব্যবহার করবো। উদাহরণ দেখলেই বিষয়টি পরিষ্কার হবে।

আমরা এখন একটি প্রোগ্রাম লিখবো, যেটি একটি সংখ্যার বর্গ বের করবে। প্রোগ্রামটি প্রতিবার ব্যবহারকারির কাছ থেকে ইনপুট চাইবে আর তার বর্গ প্রিন্ট করবে। তবে ইনপুটের সংখ্যাটি যদি 0 হয়, তখন প্রোগ্রামটি বন্ধ হয়ে যাবে।

```
while True:
    n = input("Please enter a number (0 to exit): ")
    n = int(n)
    if n == 0:
        break
    print("Square of", n, "is", n*n)
```

পরের প্রোগ্রামটি আমরা sqrn.py নামের ফাইলে সেভ করে রান করতে পারি। এই প্রোগ্রামে আমি while লুপের ভেতরে শর্ত ব্যবহার করেছি True, যা সবসময়ই সত্যি। তাহলে লুপটি চলতেই থাকবে, কখনো বন্ধ হবে না। বন্ধ করার বা লুপ থেকে বের হওয়ার উপায় হচ্ছে break। লুপের ভেতরে প্রতিবার আমরা শুরুতে একটি সংখ্যা ইনপুট নিচ্ছি। সেই সংখ্যার মান 0 কি না, সেটি পরীক্ষা করছি। পরীক্ষার ফলাফল সত্য হলে আমরা লুপ থেকে বের হয়ে যাচ্ছি। নইলে লুপের কাজ যথারীতি চলবে, অর্থাৎ print() ফাংশনের কাজ সম্পন্ন হবে, তারপর আবার লুপের ভেতরের প্রথম লাইন থেকে কাজ চলতে থাকবে।

ওপরের প্রোগ্রামটি কিন্তু ধনাত্মক ও ঋণাত্মক – দুই ধরণের সংখ্যার জন্যই কাজ করবে। এখন আমরা যদি চাই যে, কেবল ধনাত্মক সংখ্যার বর্গ দেখাবো, তাহলে প্রোগ্রামটি নিচের মতো করে পরিবর্তন করে নিতে পারি।

```
while True:
    n = input("Please enter a positive number (0 to exit): ")
    n = int(n)
    if n < 0:
        print("We only allow positive number. Please try again.")
        continue
    if n == 0:
        break
    print("Square of", n, "is", n*n)
```

এখানে n < 0 শর্তটি সত্যি হলে আমরা একটি বার্তা প্রিন্ট করে লুপটি আবার চালাবো। এজন্য continue ব্যবহার করা হয়েছে। প্রোগ্রামটি রান করে বিভিন্ন রকম ইনপুট দিলেই বিষয়টি পরিষ্কার হবে। যারা নিজেরা প্রোগ্রাম করে বিষয়টি পরীক্ষা-নিরীক্ষা করবে না, তাদের আসলে প্রোগ্রামিং শেখা হবে না।

এখন আমরা আরেকটি প্রোগ্রাম লিখবো। এটি লেখার উদ্দেশ্য হচ্ছে break ও continue-এর আরো একটি ব্যবহার দেখানো। প্রোগ্রামটির কাজ কী, সেটি কোড দেখে, নিজে প্রোগ্রাম লিখে রান করে ও সেই সঙ্গে চিন্তা করে পুরোপুরি বুঝতে পারা যাবে। কারো কারো কাছে প্রোগ্রামটি একটু জটিল মনে হতে পারে, তবে বইতে এখন পর্যন্ত যা দেখানো হয়েছে, প্রোগ্রামটি বুঝার জন্য সেই জ্ঞানই যথেষ্ট।

```
terminate_program = False
while not terminate_program:
    number1 = input("Please enter a number: ")
    number1 = int(number1)
    number2 = input("Please enter another number: ")
    number2 = int(number2)

    while True:
        operation = input("Please enter add/sub or quit to exit: ")
        if operation == "quit":
            terminate_program = True
            break
        if operation not in ["add", "sub"]:
            print("Unknown operation!")
            continue
        if operation == "add":
            print("Result is", number1 + number2)
            break
        if operation == "sub":
            print("Result is", number1 - number2)
            break
```

Published by subeen on April 16, 2018

ফাংশনের সঙ্গে পরিচয় আমাদের অনেক আগেই হয়ে গিয়েছে। প্রথমেই যখন Hello world প্রিন্ট করেছিলাম, তখনই আমরা print() ফাংশন ব্যবহার করেছি। এছাড়াও আরো অনেক ফাংশন ব্যবহার করেছি, যেমন input(), len(), type() ইত্যাদি। আবার আমরা turtle মডিউল যখন ব্যবহার করেছি, তখন টার্টলের অনেক ফাংশনও ব্যবহার করেছি, যেমন forward(), left(), right(), dot() ইত্যাদি। ফাংশনগুলো কী কাজ করে, এটি আমাদের জানতে হবে, কিন্তু কিভাবে কাজ করে, সেটি এখন আমাদের জানার প্রয়োজন নেই। এসব ফাংশন তৈরি করে দেওয়া না থাকলে আমাদেরকে অনেক বেশি কষ্ট করতে হতো। যেমন print() ফাংশনটি কিভাবে কাজ করবে, সেটি পাইথনের বিল্টইনস্ মডিউলে লেখা আছে (তবে এই মডিউলটি আমাদের ইমপোর্ট করতে হয় না)। তেমনি dot() কিভাবে কাজ করবে, সেটি লেখা আছে টার্টল মডিউলের ভেতরে। এই অধ্যায়ে আমরা দেখবো, কিভাবে ফাংশন তৈরি করতে হয়, কিভাবে মডিউল তৈরি করতে হয় এবং প্রয়োজনীয় কিছু ফাংশনের ব্যবহার।

আমরা কখন ফাংশন তৈরি করবো? যখন আমাদেরকে একটি নির্দিষ্ট কাজ করতে হবে এবং সেই কাজটি একাধিকবার করার প্রয়োজন হবে, তখন আমরা সেই কাজের জন্য একটি ফাংশন তৈরি করে ফেলবো। আবার কখনও কখনও একটি কাজ একবার করলেও আমরা সেটির জন্য আলাদা ফাংশন তৈরি করতে পারি, যেন কোড বুঝতে সহজ হয়। যেমন ধরা যাক, আমাকে কোনো প্রোগ্রামে অনেকবার দুটি সংখ্যা যোগ করতে হবে। তখন আমরা এই যোগ করার জন্য একটি ফাংশন তৈরি করে ফেলতে পারি এভাবে :

```
def add(n1, n2):
    return n1 + n2
```

আমরা দেখতে পাচ্ছি যে, ফাংশনের শুরুতে লিখতে হবে def, তাহলে পাইথন বুঝবে যে এখানে একটি ফাংশন তৈরি করা হচ্ছে (বা সংজ্ঞায়িত করা হচ্ছে, definition শব্দের প্রথম তিন অক্ষর def)। তারপর ফাংশনের নাম দিতে হবে। আমরা নাম দিয়েছি add। ফাংশনের নাম দেখে যেন বোঝা যায় যে, ফাংশনটি কী কাজ করবে। তারপরে প্রথম বন্ধনীর ভেতরে ফাংশনের প্যারামিটার লিখতে হবে। সব ফাংশনে প্যারামিটার থাকে না। প্যারামিটার থাকবে কী না এবং কয়টি, সেটি নির্ভর করে আমরা কী ফাংশন তৈরি করছি, তার ওপর। যেমন এখানে আমি তৈরি করছি দুটি সংখ্যার যোগফল বের করার ফাংশন। তাহলে তো আমাকে কোন দুটি সংখ্যা আমি যোগ করবো, সেগুলো জানতে হবে বা ইনপুট নিতে হবে। ফাংশনের ক্ষেত্রে প্যারামিটার হচ্ছে ইনপুট নেওয়ার পদ্ধতি। তারপরে একটি কোলন চিহ্ন দিতে হবে। পরের লাইন থেকে ফাংশনের ভেতরের কোড লিখতে হবে এবং সেগুলো ইনডেনটেশন করা থাকতে হবে, নইলে পাইথনের পক্ষে বোঝা সম্ভব হবে না যে কোন অংশ ফাংশনের ভেতরে আর কোন অংশ বাইরে। ফাংশন থেকে আবার এক বা একাধিক জিনিস ফেরত পাঠানো যায়, যাকে বলে রিটার্ন করা। আমাদের যেমন যোগফল ফেরত পাঠানো দরকার, তাই আমরা n1 + n2 এর মান রিটার্ন করছি। আমরা এখন ফাংশনটির ব্যবহার

দেখবো। আমাদের কোনো নিয়মকানুন মুখস্থ করার প্রয়োজন নেই, চর্চা করে ও চিন্তা করে আমরা প্রোগ্রামিং শিখবো।

```
>>> def add(n1, n2):
...     return n1 + n2
...
>>> n = 10
>>> m = 5
>>> result = add(n, m)
>>> print(result)
15
>>>
>>> number1 = 10
>>> number2 = 10
>>> result = add(number1, number2)
>>> print(result)
20
>>>
>>> n1 = 20
>>> n2 = 10
>>> print(add(n1, n2))
30
>>> print(add(2.5, 3.5))
6.0
>>>
```

আমরা ফাংশনটির বিভিন্ন রকম ব্যবহার দেখলাম। যেসব বিষয় খেয়াল করতে হবে, সেগুলো হচ্ছে –

- যদিও ফাংশনের প্যারামিটার হিসেবে n1 ও n2 ব্যবহার করা হয়েছে, আমরা কিন্তু ফাংশন কল করার সময় যেকোনো নামের আর্গুমেন্ট ব্যবহার করতে পারি।
- আমরা যখন ফাংশনটি সংজ্ঞায়িত বা ডিফাইন করছি বা সহজ বাংলায় ফাংশনটি তৈরি করছি, তখন যে n1, n2 লিখলাম, এগুলোকে বলে ফাংশনের প্যারামিটার (parameter)। আর আমরা যখন ফাংশনটি কল করছি, যেমন add(n, m) বা add(n1, n2) বা add(2, 3) এখানে n, m বা n1, n2 বা 2, 3 হচ্ছে আর্গুমেন্ট (argument)। নামগুলো মুখস্থ করতে হবে না, তবে জেনে রাখা ভালো। ভবিষ্যতে ইংরেজি বই পড়ার সময় কাজে লাগবে।
- আর্গুমেন্ট হিসেবে ভ্যারিয়েবল ব্যবহার করা যায়, আবার সরাসরি বিভিন্ন সংখ্যাও ব্যবহার করা যায়।

ফাংশনের যে সবসময় প্যারামিটার থাকতেই হবে, সেরকম কোনো কথা নেই। আবার ফাংশন থেকে যে কোনো কিছু রিটার্ন করতে হবে, এমন কোনো কথা নেই। আমরা ফাংশন থেকে কোনো কিছু রিটার্ন না করলে পাইথন আপনাআপনি None রিটার্ন করে। এটি পরীক্ষা করে দেখা যেতে পারে। আবার অনেকসময় ফাংশন থেকে একাধিক জিনিসও রিটার্ন করতে হতে পারে। সেক্ষেত্রে যেসব জিনিস রিটার্ন করতে হবে, return স্টেটমেন্টের পর সেগুলো কমা দিয়ে পৃথক করে দিতে হবে। যেমন : return a, b, c।

আমরা টার্টল ব্যবহার করে বর্গক্ষেত্র আঁকার একটি প্রোগ্রাম তৈরি করেছিলাম। প্রোগ্রামটিতে আমরা বর্গক্ষেত্র আঁকার কাজটি একটি ফাংশন তৈরি করে করতে পারি।

```
def draw_square(side_length):
    for i in range(4):
        turtle.forward(side_length)
        turtle.left(90)
```

তাহলে একটি সম্পূর্ণ প্রোগ্রাম লিখে দেখি, সেই বৃত্ত তৈরির প্রোগ্রামটি। নিচের প্রোগ্রামটি সাদাকালো, তবে কেউ চাইলে বিভিন্ন রং ব্যবহার করতে পারে :

```
import turtle

def draw_square(side_length):
    for i in range(4):
        turtle.forward(side_length)
        turtle.left(90)

counter = 0
while counter < 90:
    draw_square(100)
    turtle.right(4)
    counter += 1

turtle.exitonclick()
```

অনুশীলনী: একটি ফাংশন তৈরি করতে হবে যা প্যারামিটার হিসেবে একটি বাহুর দৈর্ঘ্য নেবে এবং একটি সমবাহু ত্রিভূজ আঁকবে।

এখন আমরা কিছু উদাহরণের মাধ্যমে ফাংশন সম্পর্কে আরো জানবো।

```
def myfnc(x):
    print("inside myfnc", x)
    x = 10
    print("inside myfnc", x)

x = 20
myfnc(x)
print(x)
```

ওপরের প্রোগ্রামটি রান করলে আমরা আউটপুট পাবো এমন :

```
inside myfnc 20
inside myfnc 10
20
```

তাহলে আমরা দেখতে পেলাম, ভ্যারিয়েবলের নাম যদিও একই, কিন্তু myfnc-এর ভেতরে x-এর মান পরিবর্তন করে দিলেও ফাংশনের বাইরে x-এর মান পরিবর্তিত হয় নি। এর কারণ হচ্ছে myfnc যখন কল হয়, তখন সে আর্গুমেন্ট হিসেবে যেসব ভ্যারিয়েবল পায়, সেগুলোর একটা কপি তৈরি হয়। তাই myfnc-এর x আর তার বাইরের x আসলে দুটি আলাদা ভ্যারিয়েবল। একটি ফাংশনের ভেতরে যেসব ভ্যারিয়েবল তৈরি করা হয় সেগুলো হচ্ছে ওই ফাংশনের লোকাল (local) ভ্যারিয়েবল। ফাংশনের বাইরে তার অস্তিত্ব থাকে না।

ফাংশনের বাইরে যদি কোনো ভ্যারিয়েবল থাকে, তাহলে ফাংশনের ভেতর থেকে ওই ভ্যারিয়েবল পাওয়া যায়। একে বলে গ্লোবাল (global) ভ্যারিয়েবল। নিচের প্রোগ্রাম রান করলে সেটি আমরা দেখতে পাবো :

```
def myfnc(y):
    print("y =", y)
    print("x =", x)

x = 20
myfnc(x)
```

প্রোগ্রামের আউটপুট হবে এরকম :

```
y = 20
x = 20
```

এখন আমরা যদি myfnc এর বাইরে থেকে y-এর মান দেখতে চাই, সেটি কি সম্ভব? নিচের প্রোগ্রাম রান করলেই বুঝতে পারা যাবে।

```
def myfnc(y):
    print("y =", y)
    print("x =", x)

x = 20
myfnc(x)
print("y:", y)
```

আউটপুট হবে এরকম :

```
y = 20
x = 20
Traceback (most recent call last):
  File "fnc_test.py", line 7, in <module>
    print("y:", y)
NameError: name 'y' is not defined
```

অর্থাৎ y নামে কোনো কিছু পাওয়া যায় নি।

এখন আমরা দেখবো, ফাংশনের প্যারামিটারের ডিফল্ট (default) মান কিভাবে দেওয়া যায়।

```
def myfnc(y=10):
    print("y =", y)

x = 20
myfnc(x)
myfnc()
```

এই প্রোগ্রামে আমি দুইবার myfnc ফাংশন কল করলাম। প্রথমবার আর্গুমেন্ট হিসেবে x পাঠাচ্ছি। দ্বিতীয়বার কিছুই পাঠাচ্ছি না। কিন্তু ফাংশনের প্যারামিটারে আবার বলে দিয়েছি y=10। এর মানে হচ্ছে যদি কোনো আর্গুমেন্ট পাঠানো না হয়, তাহলে y-এর মান হবে 10, আর যদি কোনো আর্গুমেন্ট পাঠানো হয়, তাহলে আর্গুমেন্টে যেই ভ্যারিয়েবলটি পাঠানো হলো, সেই ভ্যারিয়েবলের মান y-তে কপি হবে।

এবারে আরেকটি প্রোগ্রাম লিখবো :

```
def myfnc(x, y=10, z):
    print("x =", x, "y =", y, "z =", z)

x = 5
y = 6
z = 7
myfnc(x, y, z)
```

ওপরের কোড যদি রান করি, তাহলে আউটপুট পাবো এরকম :

```
    def myfnc(x, y=10, z):
             ^
SyntaxError: non-default argument follows default argument
```

এই এররের অর্থ হচ্ছে আমরা যদি ডিফল্ট মান দেই, তাহলে তারপরে সবগুলো আর্গুমেন্টে ডিফল্ট মান থাকতে হবে। মানে আমরা যদি লিখতাম def myfnc(x, y=10, z=0) তাহলে এই এরর আর আসবে না। আমরা প্রোগ্রাম লিখে পরীক্ষা করে দেখি। নিচের প্রোগ্রামটি রান করে আউটপুট দেখে বুঝে নিতে হবে।

```
def myfnc(x, y=10, z=0):
    print("x =", x, "y =", y, "z =", z)

x = 5
y = 6
z = 7
myfnc(x, y, z)
myfnc(x, y)
myfnc(x)
```

আর আমরা যদি চাই যে, z-এর কোনো ডিফল্ট মান থাকবে না তাহলে এভাবে লিখতে হবে : myfnc(x, z, y = 10)। কিন্তু এভাবে লিখলে তো একটা সমস্যা আছে, দ্বিতীয় আর্গুমেন্ট যেটি পাঠাবো, সেটি তো z-এ আসবে, কারণ প্যারামিটারের নাম তো বিবেচনা করা হয় না, বরং প্রথমটি x-এ, দ্বিতীয়টি z-এ এবং তৃতীয়টি y-এ কপি হবে। কিন্তু পাইথনে একটি উপায় আছে যেখানে নির্দিষ্ট প্যারামিটারে নির্দিষ্ট মান পাঠানো যায়। নিচের উদাহরণে আমরা সেটি দেখবো।

```
def myfnc(x, z, y=10):
    print("x =", x, "y =", y, "z =", z)

myfnc(x = 1, y = 2, z = 5)
a = 5
b = 6
myfnc(x = a, z = b)
a = 1
b = 2
c = 3
myfnc(y = a, z = b, x = c)
```

প্রোগ্রামটি রান করলে নিচের মতো আউটপুট আসবে :

```
x = 1 y = 2 z = 5
x = 5 y = 10 z = 6
x = 3 y = 1 z = 2
```

পাইথনে সুযোগ-সুবিধা অনেক বেশি। তবে এখানে একটি কথা বলা প্রয়োজন। বইটি প্রথমবার পড়ার পরে বেশিরভাগ পাঠকেরই এত কিছু মনে থাকবে না এবং এটি খুবই স্বাভাবিক। এতে চিন্তিত হওয়ার কিছু নেই। বইটি দ্বিতীয় বা তৃতীয়বার পড়ার (এবং সেই সঙ্গে প্রতিবারই বইয়ের সব উদাহরণ ও অনুশীলনীগুলো করতে হবে) পর এগুলো মনে থাকবে।

আমরা চাইলে ফাংশনের ভেতরে লিস্টও পাঠাতে পারি। যেমন, এখন আমি একটি ফাংশন লিখবো, যেটিতে আর্গুমেন্ট হিসেবে একটি লিস্ট পাঠানো যাবে এবং আমরা সেই লিস্টের সংখ্যাগুলো যোগ করে যোগফল ফেরত পাঠাবো। যদিও লিস্টে আমরা যেকোনো কিছুই পাঠাতে পারি; কিন্তু যেহেতু যোগ করবো, তাই আমরা সংখ্যার লিস্ট পাঠাবো।

```
def add_numbers(numbers):
    result = 0
    for number in numbers:
        result += number
    return result

result = add_numbers([1, 2, 30, 4, 5, 9])
print(result)
```

এখন, ফাংশনের ভেতর ভ্যারিয়েবল পাঠানো আর লিস্ট পাঠানোর মধ্যে একটি পার্থক্য আছে। যে ফাংশনের ভেতরে লিস্ট পাঠানো হয়েছে, সেখানে যদি লিস্টটি পরিবর্তন করা হয়, তাহলে আসল লিস্টও পরিবর্তিত হয়ে যায়। এটিও আমরা একটি প্রোগ্রাম লিখে যাচাই করো নেবো। তবে তার

আগে আমাদের আরো একটি জিনিস জেনে নিতে হবে। আমরা যদি লিস্টের কোনো নির্দিষ্ট উপাদান পেতে চাই, যেমন প্রথম, দ্বিতীয়, তৃতীয় ইত্যাদি, তাহলে আমরা তৃতীয় বন্ধনীর ভেতরে সেই সংখ্যাটি উল্লেখ করে দিতে পারি। একে বলা হয় ইনডেক্স। তবে পাইথনে লিস্টের ইনডেক্স কিন্তু 1 থেকে নয়, 0 থেকে শুরু হয়। অর্থাৎ, আমরা যদি লিস্টের প্রথম উপাদান একসেস করতে চাই, তাহলে আমরা লিস্টের নামের পর [0] লিখবো। country নামের একটি লিস্টে যদি পৃথিবীর সবগুলো দেশের নাম থাকে, আর সেই লিস্টের 50-তম দেশটি আমরা দেখতে চাই, তাহলে আমাদেরকে country[49] প্রিন্ট করতে হবে।

```
def test_fnc(li):
    li[0] = 10

my_list = [1, 2, 3, 4]
print("before function call", my_list)
test_fnc(my_list)
print("after function call", my_list)
```

প্রোগ্রামটি রান করলে আমরা আউটপুট পাবো এরকম :

```
before function call [1, 2, 3, 4]
after function call [10, 2, 3, 4]
```

এরকম হওয়ার কারণ কী? কারণ আমরা যখন একটি লিস্ট অন্য একটি ভ্যারিয়েবলে অ্যাসাইন করি, তাহলে নতুন কোনো লিস্ট তৈরি হয় না, বরং নতুন ভ্যারিয়েবল আর পুরনো ভ্যারিয়েবলে একই লিস্ট থাকে। সেটিও আমরা পরীক্ষা করে দেখবো পাইথন ইন্টারপ্রেটারে।

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = list1
>>> print(list1)
[1, 2, 3, 4]
>>> print(list2)
[1, 2, 3, 4]
>>> list2[0] = 100
>>> print(list2)
[100, 2, 3, 4]
>>> print(list1)
[100, 2, 3, 4]
```

তাই ফাংশনের ভেতরে লিস্ট পাঠানোর সময় কিংবা আরেকটি ভ্যারিয়েবলে লিস্ট অ্যাসাইন করার সময় আমাদের সতর্ক হতে হবে। নইলে প্রোগ্রামে বাগ-এর সৃষ্টি হবে। কম্পিউটার প্রোগ্রামের ত্রুটিকে বলে বাগ (bug)।

আমরা যে একটু আগে লিস্টের উপাদানগুলো যোগ করার জন্য একটি ফাংশন লিখলাম, পাইথনে কিন্তু সেই কাজ করার জন্য একটি বিল্টইন ফাংশন আছে। আমাদের প্রয়োজন হলে আমরা সেই ফাংশন ব্যবহার করবো। তাহলে আমি উদাহরণের প্রোগ্রাম কেন লিখলাম? ফাংশনে কিভাবে লিস্ট পাঠাতে হয়, সেটি দেখানোর জন্য।

```
>>> li = [1, 2, 3]
>>> result = sum(li)
>>> print(result)
6
```

আমরা বইয়ের পরের অধ্যায়গুলোতে আরো বিভিন্ন বিল্টইন ফাংশন ব্যবহার করবো এবং নিজেরাও কিছু কিছু ফাংশন তৈরি করবো।

অনুশীলনী:

- একটি ফাংশন তৈরি করতে হবে, যে প্যারামিটারে একটি লিস্ট নেবে এবং লিস্টের সংখ্যাগুলোর গড় নির্ণয় করে ফেরত পাঠাবে।
- একটি নামতা প্রিন্ট করার ফাংশন তৈরি করতে হবে। ফাংশনে কোনো সংখ্যা পাঠালে সেই সংখ্যার নামতা প্রিন্ট করবে। আর কোনো কিছু না পাঠালে 1-এর নামতা প্রিন্ট করবে।

বিশেষ দ্রষ্টব্য – ফাংশন বিষয়ে বাংলা ভিডিও লেকচার দেখা যাবে এখানে – https://goo.gl/ozYWtv

উল্লেখ্য যে, ভিডিও লেকচারটি পাইথন ২ এর জন্য তৈরি করা হলেও, বেশিরভাগ প্রোগ্রামই পাইথন ৩-এও চলবে। আর, যেখানে print স্টেটম্যান্ট ব্যবহার করা হয়েছে, সেখানে print() ব্যবহার করতে হবে।

স্ট্রিং নিয়ে কাজকারবার

Published by subeen on April 15, 2018

স্ট্রিং (string)-এর সঙ্গে আমরা ইতিমধ্যেই পরিচিত হয়েছি। এই অধ্যায়ে আমরা স্ট্রিং-এর আরো ব্যবহার শিখবো। বেশিরভাগ ক্ষেত্রেই আমরা আলাদা ফাইলে প্রোগ্রাম না লিখে ইন্টারপ্রেটার ব্যবহার করবো, যাতে আমরা উদাহরণগুলো নিজেরা দ্রুত অনুসরণ করতে পারি।

শূণ্য বা তারচেয়ে বেশি সংখ্যক অক্ষর বা চিহ্ন দিয়ে স্ট্রিং তৈরি হয়। স্ট্রিং শুরু ও শেষ করতে হয় ডবল কোটেশন অথবা সিঙ্গেল কোটেশন চিহ্ন ব্যবহার করে। স্ট্রিংয়ে মোট কয়টি অক্ষর আছে, সেটি আমরা বের করতে পারি len() ফাংশন ব্যবহার করে।

```
>>> s = "hello"
>>> len(s)
5
>>> l = len(s)
>>> l
5
>>> print(l)
5
>>>
>>> s = ''
>>> len(s)
0
>>>
>>> s = ""
>>> len(s)
0
>>>
```

আবার আমাদের যদি এমন কিছু প্রিন্ট করতে হয়, যেখানে সিঙ্গেল কোটেশন আছে (যেমন Dimik's), তখন আমরা সেটি দুইভাবে করতে পারি :

```
>>> s = "Dimik's"
>>> print(s)
Dimik's
>>> s = 'Dimik\'s'
>>> print(s)
Dimik's
>>>
```

একটি স্ট্রিংকে যদিও আমরা একটি ভ্যারিয়েবলে রাখি, আমরা চাইলে প্রতিটি অক্ষর আলাদাভাবে পেতে পারি। s যদি একটি স্ট্রিং হয়, তাহলে প্রথম অক্ষর পাওয়া যাবে s[0]-তে, দ্বিতীয় অক্ষর পাওয়া যাবে s[1]-এ।

```
>>> country = "Bangladesh"
```

```
>>> country[0]
'B'
>>> country[1]
'a'
>>> country[2]
'n'
>>> country[6]
'd'
>>> country[8]
's'
>>> country[9]
'h'
>>> country[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

ওপরের উদাহরণ থেকে আমরা বুঝতে পারছি, স্ট্রিংয়ের দৈর্ঘ্য যদি len হয়, তাহলে প্রথম অক্ষরটি থাকবে 0 ইনডেক্সে আর শেষ অক্ষরটি থাকবে len-1 ইনডেক্সে। আমরা স্ট্রিংয়ের ওপর লুপও চালাতে পারি এভাবে :

```
>>> for c in country:
...     print(c)
...
B
a
n
g
l
a
d
e
s
h
>>>
```

অনেকটা লিস্টের মতোই। কিন্তু লিস্টের সঙ্গে একটি পার্থক্য হচ্ছে, আমরা চাইলে লিস্টের কোনো উপাদান পরিবর্তন করতে পারি, স্ট্রিংয়ের ক্ষেত্রে সেটি সম্ভব নয়।

```
>>> c = ['A', 'b', 'c']
>>> print(c)
['A', 'b', 'c']
>>> c[0] = 'a'
>>> print(c)
['a', 'b', 'c']
>>>
>>> country = "Bangladesh"
>>> country[0] = 'b'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

এজন্য পাইথনের ইংরেজি বইগুলোতে স্ট্রিংকে বলে ইমমিউটেবল (immutable) বা নন-মিউটেবল (non-mutable), কারণ এর পরিবর্তন করা যায় না।

দুটি স্ট্রিং জোড়া লাগানোর জন্য আমরা যোগ চিহ্ন ব্যবহার করতে পারি।

```
>>> country = "Bangla" + "desh"
>>> print(country)
Bangladesh
>>> x = "50" + "5"
>>> print(x)
505
>>>
```

এই যোগের কাজটা করতে গিয়ে আমরা অনেকসময় গড়বড় করে ফেলি। তাই একটু সতর্কতা অবলম্বন করতে হবে যে, আমরা আসলে সংখ্যা যোগ করছি নাকি স্ট্রিং জোড়া লাগাচ্ছি।

অনেকসময় আমাদেরকে একটি স্ট্রিংয়ের ভেতরে বিশেষ একটি স্ট্রিং খুঁজে বের করতে হবে। তখন আমরা find() মেথড ব্যবহার করবো। যেমন :

```
>>> country = "Bangladesh"
>>> country.find("Ban")
0
>>> country.find("ang")
1
>>> country.find("Bangla")
0
>>> country.find("Bengla")
-1
>>> country.find("desh")
6
```

এখানে আমরা country (অর্থাৎ Bangladesh)-এর ভেতরে বিভিন্ন স্ট্রিং খুঁজে বের করার চেষ্টা করেছি। যদি আমরা যা খুঁজছি, সেটি পাওয়া না যায়, তখন find() মেথড -1 রিটার্ন করে। আর যদি খুঁজে পাওয়া যায়, তাহলে যেই পজিশনে সেটি খুঁজে পাওয়া গিয়েছে, সেই পজিশন বা ইনডেক্স রিটার্ন করে।

আমরা যদি কোনো একটি স্ট্রিংয়ের ভেতরে কোনো কিছু বদলে দিতে চাই, তাহলে আমরা replace() মেথড ব্যবহার করে সেটি করতে পারি। যেমন :

```
>>> country = "North Korea"
>>> new_country = country.replace("North", "South")
>>> print(new_country)
South Korea
>>> print(country)
North Korea
>>>
>>> text = "this is a test. this is another test. this is final test."
```

```
>>> new_text = text.replace("this", "This")
>>> print(new_text)
This is a test. This is another test. This is final test.
>>> print(text)
this is a test. this is another test. this is final test.
```

ওপরের উদাহরণে আমরা দেখলাম যে, replace() মেথডটি নতুন একটি স্ট্রিং রিটার্ন করে, কিন্তু মূল স্ট্রিংয়ে কোনো পরিবর্তন করে না। তবে আমরা চাইলে আগের স্ট্রিং যেই ভ্যারিয়েবলে রেখেছিলাম, নতুন স্ট্রিংটিও একই ভ্যারিয়েবলে রাখতে পারি। সেক্ষেত্রে একই নামের একটি নতুন ভ্যারিয়েবলে নতুন স্ট্রিং অ্যাসাইন হবে। যদিও আমাদের মনে হতে পারে আগের স্ট্রিং পরিবর্তিত হয়েছে, সেটি আসলে সত্যি নয়, কারণ স্ট্রিং পরিবর্তন করা যায় না, একটু আগেই আমরা সেটি দেখেছি।

```
>>> text = "hello"
>>> text = text.replace("hello", "Hello")
>>> print(text)
Hello
```

ওপরে ঘটনা যা ঘটছে, তা হলো, text.replace(“hello”, “Hello”) একটি নতুন স্ট্রিং তৈরি করছে, আর সেটিকে আমরা text ভ্যারিয়েবলেই রাখছি। “hello” স্ট্রিংটি বদলায়নি, বরং নতুন একটি স্ট্রিং তৈরি হয়েছে – এই বিষয়টি খেয়াল রাখতে হবে। প্রোগ্রামিংয়ে যারা নতুন, তাদের হয়ত বুঝতে একটু সমস্যা হতেও পারে, তাতে কোনো অসুবিধা নেই।

স্ট্রিংয়ের শুরুতে ও শেষে অনেক সময় স্পেস ক্যারেক্টার থাকে, যেগুলো আমাদের বাদ দেওয়ার প্রয়োজন হতে পারে। সেজন্য আমরা strip(), lstrip() ও rstrip() মেথড ব্যবহার করতে পারি। lstrip() মেথড স্ট্রিংয়ের বাঁ দিকের স্পেসগুলো বাদ দেবে, rstrip() মেথড ডানদিকের স্পেসগুলো বাদ দেবে, আর strip() মেথড দুইদিকের স্পেসগুলোই বাদ দেবে।

```
>>> text = " this is a string. "
>>> text
' this is a string. '
>>> text.lstrip()
'this is a string. '
>>> text.rstrip()
' this is a string.'
>>> text.strip()
'this is a string.'
>>> text
' this is a string. '
>>>
```

ওপরের উদাহরণে আমরা আরো দেখলাম যে text-এর কিন্তু কোনো পরিবর্তন হলো না। কারণ মেথডগুলো আসলে নতুন একটি স্ট্রিং রিটার্ন করে। তাই সেগুলো পেতে চাইলে নতুন কোনো ভ্যারিয়েবলে (কিংবা text ভ্যারিয়েবলে) রাখতে হবে।

```
>>> text = " this is a string. "
>>> new_text = text.rstrip()
```

```
>>> new_text
' this is a string.'
>>> text
' this is a string. '
>>>
```

আবার আমরা যদি কোনো স্ট্রিংকে আপারকেস (uppercase), লোয়ারকেস (lowercase)-এ রূপান্তর করতে চাই, কিংবা কেবল প্রথম অক্ষরটি বড় হাতের করে দিতে চাই, সেজন্যও ফাংশন তৈরি করা আছে।

```
>>> s1 = "Bangladesh"
>>> s_up = s1.upper()
>>> s_up
'BANGLADESH'
>>> s_lo = s1.lower()
>>> s_lo
'bangladesh'
>>> s = "hello"
>>> s_cap = s.capitalize()
>>> s_cap
'Hello'
```

কোনো স্ট্রিংয়ে যদি অনেকগুলো ছোট স্ট্রিং এক বা একাধিক স্পেস দিয়ে আলাদা করা থাকে আর আমরা কেবল সেই ছোট স্ট্রিংগুলো পেতে চাই, তাহলে আমরা split() মেথড ব্যবহার করে সেগুল একটি লিস্টে পেতে পারি।

```
>>> str = "I   am  a programmer."
>>> words = str.split()
>>> print(words)
['I', 'am', 'a', 'programmer.']
>>> words
['I', 'am', 'a', 'programmer.']
>>> for word in words:
...     print(word)
...
I
am
a
programmer.
>>>
```

স্ট্রিংয়ের ভেতরের স্ট্রিংকে সাবস্ট্রিং (sub-string)ও বলে। সেটের যেমন সাবসেট। এখন কোনো স্ট্রিংয়ের ভেতরে একটি স্ট্রিং কতবার আছে, সেটি গণনা করারও একটি সহজ উপায় আছে, count() মেথডের মাধ্যমে।

```
>>> str = "This is"
>>> str.count("is")
2
```

কখনো কখনো আমাদের জানার দরকার হতে পারে যে, একটি স্ট্রিংয়ের শুরুতে কিংবা শেষে একটি স্ট্রিং (বা সাবস্ট্রিং) আছে কী নেই। সেজন্য পাইথনে দুটি মেথড তৈরি করা আছে, startswith() ও endswith()। নিচের উদাহরণ দেখলেই বুঝতে পারবো।

```
>>> s = "Bangladesh"
>>> s.startswith("Ban")
True
>>> s.startswith("ban")
False
>>> s.startswith("an")
False
>>> s.endswith("Ban")
False
>>> s.endswith("desh")
True
>>> s.endswith("h")
True
```

আমাদের হয়ত এমন লজিকের দরকার হতে পারে যে কারো নাম Mr. দিয়ে শুরু হলে আমরা তাকে Dear Sir বলে সম্বোধন করবো। তখন এরকম কোড লেখা যেতে পারে :

```
>>> name = "Mr. Anderson"
>>> if name.startswith("Mr."):
...     print("Dear Sir")
...
Dear Sir
```

ওপরের প্রোগ্রামটিতে ইনপুট নেওয়ার জন্য আমরা টেক্সটবক্সও ব্যবহার করতে পারি। নিচে টার্টল মডিউল ব্যবহার করে আমরা টেক্সট ইনপুট নিলাম। একটি প্রোগ্রাম অনেকভাবেই লেখা যায়, এটি তার একটি ভালো উদাহরণ।

```
import turtle

name = turtle.textinput("name", "What is your name?")
name = name.lower()
if name.startswith("mr"):
    print("Hello Sir, how are you?")
elif name.startswith("mrs") or name.startswith("miss") or
name.startswith("ms"):
    print("Hello Madam, how are you?")
else:
    name = name.capitalize()
    str = "Hi " + name + "! How are you?"
    print(str)

turtle.exitonclick()
```

এতক্ষণ স্ট্রিংয়ের নানান রকমের কাজ দেখলাম। ভবিষ্যতে বিভিন্ন প্রোগ্রাম তৈরি করার সময় এগুলো কাজে লাগবে। আর বইতে যা দেখানো হয়েছে, তার বাইরেও অনেক কিছু আছে, সেগুলো

আগামী দিনগুলোতে শিখে নিতে হবে। প্রোগ্রামিং নিয়ে তাড়াহুড়া করার কিছু নেই। প্রচুর সময়, ধৈর্য্য, চর্চা ও লেখাপড়া করার বিনিময়ে ভালো প্রোগ্রামার হওয়া সম্ভব। এখন একটি মজার প্রোগ্রাম দিয়ে এই অধ্যায়ের ইতি টানবো।

```
str = "a quick brown fox jumps over the lazy dog"
for c in "abcdefghijklmnopqrstuvwxyz":
    print(c, str.count(c))
```

প্রোগ্রামটি রান করে আউটপুট দেখে বুঝতে হবে কেন এটি একটি মজার প্রোগ্রাম।

অনুশীলনী:

- একটি প্রোগ্রাম লিখতে হবে, যা ইনপুট হিসেবে একটি স্ট্রিং নেবে এবং সেই স্ট্রিং থেকে মোট চারটি স্ট্রিং তৈরি করে স্ক্রিনে প্রিন্ট করবে। প্রথম স্ট্রিংটি হবে কেবল বড় হাতের অক্ষরগুলো নিয়ে, দ্বিতীয় স্ট্রিং হবে কেবল ছোট হাতের অক্ষরগুলো নিয়ে, তৃতীয় স্ট্রিং হবে কেবল অংক (digit) নিয়ে আর বাকী সবকিছু চতুর্থ স্ট্রিংয়ে থাকবে। যেমন, ইনপুট যদি Hello Test! 123 123, good. হয়, তাহলে আউটপুট হবে এমন –
- `HT`
- `elloestgood`
- `123123`
- `  !  ,  .`
- একটি প্রোগ্রাম লিখতে হবে, যা ইনপুট হিসেবে একটি স্ট্রিং নেবে এবং প্রথম ও দ্বিতীয়, তৃতীয় ও চতুর্থ, পঞ্চম ও ষষ্ঠ – এভাবে পাশাপাশি দুটি অক্ষরকে অদল-বদল করে একটি নতুন স্ট্রিং তৈরি করবে। যেমন, ইনপুট যদি হয় Bangladesh, তাহলে আউটপুট হবে aBgnaledhs।
- কোনো শব্দকে উল্টো করে সাজালেও যদি আগের মতোই শব্দ পাওয়া যায়, তাকে বলে প্যালিনড্রোম ()। যেমন: madam শব্দটির অক্ষরগুলো উল্টো করে সাজালে আমরা madam পাই। তাই madam একটি প্যালিনড্রোম। আবার toyota শব্দটির অক্ষরগুলো উল্টো করে সাজালে পাওয়া যায় atoyot, তাই এটি প্যালিন্ড্রোম নয়। এখন, একটি প্রোগ্রাম লিখতে হবে, যেটি ব্যবহারকারীর কাছ থেকে একটি স্ট্রিং ইনপুট নেবে এবং স্ট্রিংটি প্যালিনড্রোম কী না, সেটি বলে দেবে।

পাইথনের ডেটা স্ট্রাকচার – লিস্ট ও টাপল

Published by subeen on April 14, 2018

আমরা বইতে এতক্ষণ ছোট ছোট অনেক প্রোগ্রাম লিখেছি, যেগুলো প্রোগ্রামিংয়ের খুবই মৌলিক বিষয়। তবে একথাও ঠিক যে এসব বিষয় সহজ হলেও আয়ত্বে আনা খুব সহজ কোনো ব্যাপার নয়। প্রোগ্রামিং শেখার শুরুর দিকে কয়েকমাস প্রতিদিন একনাগাড়ে পাঁচ-ছয় ঘণ্টা চর্চা করা জরুরী। আর প্রোগ্রামিং যেহেতু একটা দক্ষতা, তাই বই পড়ার সঙ্গে সঙ্গে প্র্যাকটিস করাটা জরুরী।

এখন আমরা যখন একটু বড় প্রোগ্রাম লিখতে যাবো, তখন আমাদেরকে অনেক ডেটা নিয়ে কাজ করতে হবে। আমরা দেখেছি কিভাবে একটি লিস্টে অনেক ডেটা রাখা যায়। এই অধ্যায়ে লিস্ট সম্পর্কে আমরা আরো জানবো। এছাড়া পাইথনের আরো তিনটি গুরুত্বপূর্ণ ডেটা স্ট্রাকচার – টাপল, সেট ও ডিকশনারি সম্পর্কেও জানবো। এগুলোকে ডেটা স্ট্রাকচার বলে কেন? কারণ এগুলো ব্যবহার করে একটি নির্দিষ্ট উপায়ে বা বিন্যাসে ডেটা রাখা যায়। যখনই আমরা একাধিক ডেটা নিয়ে কাজ করবো, তখনই আমাদের ডেটা স্ট্রাকচার ব্যবহার করার প্রয়োজন পড়বে। যদিও এই বইতে আমরা খুব বড় কোনো প্রোগ্রাম লিখবো না, কিন্তু এসব ডেটা স্ট্রাকচার আয়ত্বে থাকাটা খুব দরকার, যেন প্রয়োজনের সময় সঠিক ডেটা স্ট্রাকচার ব্যবহার করতে কোনো বেগ পেতে না হয়।

লিস্ট (list)

লিস্ট যেহেতু আমরা পূর্বেই ব্যবহার করেছি, তাই লিস্ট নিয়ে মৌলিক কোনো আলোচনায় যাবো না। বরং লিস্টের বিভিন্ন মেথড ও তার ব্যবহার দেখবো। আর এই বইতে আমি কখনো কখনো ফাংশন, কখনো কখনো মেথড শব্দটি ব্যবহার করছি। আমরা এই বইয়ের পরবর্তি খণ্ডে ফাংশন ও মেথডের পার্থক্য বিস্তারিত জানবো। এখন এগুলো নিয়ে মাথা না ঘামালেও চলবে।

লিস্টের শেষে নতুন উপাদান যোগ করার জন্য আমরা append() মেথড ব্যবহার করবো।

```
>>> saarc = ["Bangladesh", "India", "Sri Lanka", "Pakistan", "Nepal",
"Bhutan"]
>>> print(saarc)
['Bangladesh', 'India', 'Sri Lanka', 'Pakistan', 'Nepal', 'Bhutan']
>>> saarc.append("Afganistan")
>>> print(saarc)
['Bangladesh', 'India', 'Sri Lanka', 'Pakistan', 'Nepal', 'Bhutan',
'Afganistan']
```

ওপরের উদাহরণে আমরা দেখলাম যে সার্কভুক্ত দেশগুলোর তালিকায় কিভাবে আমরা একটি নতুন দেশকে যুক্ত করতে পারি।

আমরা যখন স্ট্রিংয়ে বিভিন্ন অপারেশন চালিয়েছি কিংবা বিভিন্ন মেথড ব্যবহার করেছি, তখন কিন্তু মূল স্ট্রিংয়ে কোনো পরিবর্তন হয় নি, বরং নতুন একটি স্ট্রিং তৈরি হয়েছে। কিন্তু লিস্টের বেলায় মূল লিস্ট পরিবর্তিত হয়ে যায়। কারণ লিস্ট হচ্ছে মিউটেবল (mutable)। অর্থাৎ, লিস্ট মিউটেশন বা পরিবর্তন করা যায়।

আমরা যদি চাই, কোনো লিস্টের উপাদানগুলো একটি নির্দিষ্ট ক্রমে সাজাবো, তখন আমরা sort() মেথড ব্যবহার করতে পারি। যেমন :

```
>>> saarc.sort()
>>> print(saarc)
['Afganistan', 'Bangladesh', 'Bhutan', 'India', 'Nepal', 'Pakistan', 'Sri
Lanka']
>>> li = [1, 3, 7, 2, 4, 6, 1]
>>> li.sort()
>>> print(li)
[1, 1, 2, 3, 4, 6, 7]
>>>
```

লিস্টের উপাদানগুলো উল্টো ক্রমে নিয়ে আসতে চাইল reverse() মেথড ব্যবহার করা যায়।

```
>>> li = [1, 2, 3, 4]
>>> li.reverse()
>>> print(li)
[4, 3, 2, 1]
>>> li = ["mango", "banana", "orange"]
>>> li.reverse()
>>> print(li)
['orange', 'banana', 'mango']
>>>
```

আমরা দেখেছি যে, append() ব্যবহার করে লিস্টের শেষে নতুন উপাদান যোগ করা যায়। কিন্তু আমরা যদি চাই যে লিস্টের শুরুতে কিংবা যেকোনো অবস্থানে আমরা একটি উপাদান যোগ করবো, তখন আমরা insert() মেথড ব্যবহার করতে পারি। এই মেথডে দুটো আর্গুমেন্ট পাঠাতে হবে, insert(index, item) I index হচ্ছে লিস্টের কোন ইনডেক্সে উপাদানটি থাকবে আর item হচ্ছে সেই উপাদান। নিচের উদাহরণ দেখলেই বিষয়টি পরিষ্কার হবে :

```
>>> fruits = ["mango", "banana", "orange"]
>>> fruits.insert(0, "apple")
>>> fruits
['apple', 'mango', 'banana', 'orange']
>>> fruits.insert(2, "coconut")
>>> fruits
['apple', 'mango', 'coconut', 'banana', 'orange']
>>>
```

আমরা যদি লিস্টের কোনো উপাদান বাদ দিতে চাই, তখন আমরা remove() মেথড ব্যবহার করতে পারি। যেই উপাদানটি আমরা বাদ দিতে চাই, সেটি remove() এর ভেতরে আর্গুমেন্ট আকারে পাঠাতে হবে। আর যদি ওই উপাদান লিস্টে না থাকে তাহলে এরর মেসেজ আসবে।

```
>>> fruits
['apple', 'mango', 'coconut', 'banana', 'orange']
>>> fruits.remove("coconut")
>>> fruits
['apple', 'mango', 'banana', 'orange']
>>> fruits.remove("pineapple")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
>>>
```

তাই আমরা চাইলে এখানে একটি শর্ত দিয়ে দিতে পারি, যেন এরর মেসেজ না আসে।

```
>>> fruits
['apple', 'mango', 'banana', 'orange']
>>> item = "pineapple"
>>> if item in fruits:
...     remove(item)
... else:
...     print(item, "not in list")
...
pineapple not in list
>>>
```

এখন আমরা দেখবো pop() মেথডের ব্যবহার, যেটি কোনো লিস্টের শেষ উপাদানটি রিটার্ন করে, আবার সেই সঙ্গে লিস্ট থেকে মুছে ফেলে।

```
>>> fruits
['apple', 'mango', 'banana', 'orange']
>>> item = fruits.pop()
>>> item
'orange'
>>> fruits
['apple', 'mango', 'banana']
```

আবার আমরা যদি চাইতাম যে, লিস্টের একটি নির্দিষ্ট ইনডেক্সের উপাদান রিটার্ন করবে, তাহলে pop()-এর ভেতরে সেই ইনডেক্স দিয়ে দিলেই হবে।

```
>>> fruits
['apple', 'mango', 'banana']
>>> item = fruits.pop(1)
>>> item
'mango'
>>> fruits
['apple', 'banana']
```

```
>>>
```

একটি লিস্টের শেষে যদি আরেকটি লিস্ট যুক্ত করতে চাই, তাহলে আমরা extend() মেথড ব্যবহার করতে পারি।

```
>>> li = [1, 2, 3]
>>> li2 = [3, 4, 5, 6]
>>> li.extend(li2)
>>> li
[1, 2, 3, 3, 4, 5, 6]
```

কোনো উপাদান একটি লিস্টে কতবার আছে, সেটি আমরা দেখতে পারি count() মেথড ব্যবহার করে।

```
>>> li = [1, 2, 3, 3, 4, 5, 6]
>>> li.count(3)
2
>>> li.count(5)
1
>>> li.count(10)
0
>>>
```

আমরা যদি লিস্ট থেকে কোনো উপাদান মুছে ফেলতে চাই, কিংবা সম্পূর্ণ লিস্ট মুছে ফেলতে চাই, সেটি আমরা করতে পারি del() ফাংশন ব্যবহার করে।

```
>>> li
[1, 2, 3, 3, 4, 5, 6]
>>> del(li[1])
>>> li
[1, 3, 3, 4, 5, 6]
>>> del(li[0])
>>> li
[3, 3, 4, 5, 6]
>>> del(li)
>>> li
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'li' is not defined
```

আমাদের কখনও কখনও এমন লিস্ট তৈরি করা লাগতে পারে যেখানে শুরুতে কোনো উপাদান থাকবে না, তাকে বলে ফাঁকা লিস্ট। li = [] এভাবে আমরা ফাঁকা লিস্ট তৈরি করতে পারি।

```
>>> li = []
>>> for x in range(10):
...     li.append(x)
...
>>> print(li)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

লিস্টের মধ্যে যোগ ও গুণ অপারেশন করা যায়। এগুলো করলে কী হয় নিচের উদাহরণে আমরা দেখবো :

```
>>> li1 = [1, 2, 3]
>>> li2 = [4, 5, 6]
>>> li = li1 + li2
>>> print(li)
[1, 2, 3, 4, 5, 6]
>>>
>>> li1 = [1, 2, 3]
>>> li2 = li1 * 3
>>> print(li2)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

একটি লিস্টে যদি একাধিক স্ট্রিং থাকে, তাহলে যেগুলো যুক্ত করে আমরা একটি স্ট্রিং তৈরি করতে পারি। আবার কী দিয়ে যুক্ত করবো, সেটিও আমরা নির্ধারন করে দিতে পারি। যেমন –

```
>>> li = ["a", "b", "c"]
>>> print(li)
['a', 'b', 'c']
>>> str = "".join(li)
>>> print(str)
abc
>>> str = ",".join(li)
>>> print(str)
a,b,c
>>> str = " - ".join(li)
>>> print(str)
a - b - c
```

লিস্ট কমপ্রিহেনশনস (list comprehensions)

আমরা এখন একটি মজার জিনিস দেখবো, যাকে বলে লিস্ট কমপ্রিহেনশনস। ধরা যাক, আমার একটি লিস্ট আছে যেখানে কিছু সংখ্যা আছে। আমি চাই আরেকটি লিস্ট তৈরি করতে, যেখানে প্রতিটি উপাদান হবে আগের লিস্টের উপাদানের দ্বিগুণ। তাহলে আমরা সেটি করতে পারি এভাবে :

```
>>> li = [1, 2, 3, 4]
>>> new_li = []
>>> for x in li:
...     new_li.append(2 * x)
...
>>> print(new_li)
[2, 4, 6, 8]
```

লিস্ট কমপ্রিহেনশনস ব্যবহার করে আমি কাজটি খুব সংক্ষেপে করতে পারি :

```
>>> new_li = [2 * x for x in li]
```

```
>>> new_li
[2, 4, 6, 8]
```

আমার যদি একটি সংখ্যার লিস্ট থাকে এবং সেই লিস্ট থেকে আমি কেবল জোড় সংখ্যাগুলো নিয়ে একটি লিস্ট তৈরি করতে চাই, সেই কাজটিও জন্যও আমি লিস্ট কমপ্রিহেনশনস ব্যবহার করতে পারি। যদি লিস্ট কমপ্রিহেনশনস ব্যবহার না করতাম, তাহলে এভাবে কোড লিখতাম :

```
>>> li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> even_numbers = []
>>> for x in li:
...     if x % 2 == 0:
...         even_numbers.append(x)
...
>>> print(even_numbers)
[2, 4, 6, 8, 10]
```

আর লিস্ট কমপ্রিহেনশনস ব্যবহার করে কাজটি করা যায় এভাবে :

```
even_numbers = [x for x in range(1, 11) if x % 2 == 0]
```

অনুশীলনী : একটি লিস্ট দেওয়া আছে, যেখানে একাধিক সংখ্যা আছে। আরেকটি লিস্ট তৈরি করতে হবে, যেখানে প্রতিটি উপাদান হচ্ছে আগের লিস্টের প্রতিটি উপাদানের বর্গ। অর্থাৎ, [1, 2, 3, 4] দেওয়া থাকলে [1, 4, 9, 16] তৈরি করতে হবে। লিস্ট কমপ্রিহেনশনস ব্যবহার করার চেষ্টা করতে হবে।

টাপল (Tuple)

টাপল হচ্ছে পাইথনের একটি ডেটা স্ট্রাকচার। এটি অনেকটা লিস্টের মতোই। তবে লিস্টে আমরা যেমন তৃতীয় বন্ধনী বা স্কয়ার ব্র্যাকেট ব্যবহার করতাম, টাপলের ক্ষেত্রে প্রথম বন্ধনী (একে ইংরেজীতে বলে প্যারেন্থেসিস) ব্যবহার করতে হবে।

```
>>> x = (1, 2, 3)
>>> type(x)
<class 'tuple'>
```

এখন একটি মজার বিষয় দেখি।

```
>>> x = 1, 2, 3
>>> type(x)
<class 'tuple'>
>>>
```

তারমানে প্রথম বন্ধনী না দিলেও পাইথন x-কে একটি টাপল হিসেবেই বিবেচনা করছে। এখন এভাবে একটি উপাদনের টাপল তৈরি করা সম্ভব? নিচের কোড সেই প্রশ্নের উত্তর দেবে :

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> x = 1,
>>> type(x)
<class 'tuple'>
```

আমাদের যদি কখনো খালি টাপল তৈরি করা লাগে, তখন আমরা এভাবে করতে পারি :

```
>>> t = ()
>>> type(x)
<class 'tuple'>
```

টাপলের প্রতিটি উপাদান আমরা ইনডেক্স ব্যবহার করে পেতে পারি, আর এই ইনডেক্সও 0 থেকে শুরু হয়।

```
>>> tpl = (1, 2, 3)
>>> tpl[0]
1
>>> tpl[1]
2
>>> tpl[2]
3
>>> tpl[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
```

এখন, টাপল কিন্তু মিউটেবল (mutable) নয়। অর্থাৎ, এই যে tpl নামে একটি টাপল তৈরি করলাম যার তিনটি উপাদান যথাক্রমে 1, 2 ও 3, এখন কিন্তু আমি আর পরিবর্তন করতে পারবো না, যেটি লিস্টের বেলাতে সম্ভব।

```
>>> li = [1, 2, 3]
>>> li
[1, 2, 3]
>>> li[0] = 5
>>> li
[5, 2, 3]
>>> tpl = (1, 2, 3)
>>> tpl
(1, 2, 3)
>>> tpl[0] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

আমরা অনেকগুলো ভ্যারিয়েবল একসঙ্গে প্যাক (pack) করে একটি টাপলে রাখতে পারি, আবার একটি টাপলকে আনপ্যাক (unpack) করে তার উপাদানগুলো বিভিন্ন ভ্যারিয়েবলে রাখা যায়।

```
>>> numbers = (10, 20, 30, 40)
>>> n1, n2, n3, n4 = numbers
>>> n1
10
>>> n2
20
>>> n3
30
>>> n4
40
>>> t = n3, n4
>>> t
(30, 40)
>>> type(t)
<class 'tuple'>
```

টাপলের ভেতরে আমরা বিভিন্ন জিনিস রাখতে পারি, এবং এগুলোর ওপর লুপও চালানো যায়।

```
>>> items = (1, 2, 5.5, ["a", "b", "c"], ("apple", "mango"))
>>> for item in items:
...     print(item, type(item))
...
1 <class 'int'>
2 <class 'int'>
5.5 <class 'float'>
['a', 'b', 'c'] <class 'list'>
('apple', 'mango') <class 'tuple'>
>>> items[3]
['a', 'b', 'c']
>>> items[3][0]
'a'
>>> items[4][1]
'mango'
>>> items[4]
('apple', 'mango')
```

এখন অনেকের মনেই নিশ্চয়ই প্রশ্ন জাগছে, লিস্টের ভেতরেও কি লিস্ট রাখা যায়? হ্যাঁ, তবে আমি আর সেটি দেখালাম না, নিজে নিজে পরীক্ষা করে দেখতে হবে।

টাপলের উপাদানগুলো দিয়ে সহজেই আমরা লিস্ট তৈরি করতে পারি।

```
>>> tpl = (1, 2, 3)
>>> li = list(tpl)
>>> li
[1, 2, 3]
```

পাইথনের ডেটা স্ট্রাকচার – সেট ও ডিকশনারি

Published by subeen on April 13, 2018

সেট আমরা অনেকেই পড়েছি, না পড়লে নবম-দশম শ্রেণীর গণিত বই থেকে পড়ে নিতে হবে। তবে এখনই সেটি করা দরকার নেই, সেটের মূল ধারণাগুলো আমি আলোচনা করবো।

সেট হচ্ছে বিভিন্ন জিনিসের সমাবেশ, ইংরেজিতে বলা যায় collection of items। যেমন, আমি যদি বলি যে আমার টেবিলে এখন আছে একটি ল্যাপটপ, একটি মোবাইল, একটি গ্লাস, একটি নোটবুক, একটি কলম, তাহলে এগুলোকে একটি সেট বলা যায়। সেটের ভেতরের জিনিসগুলো দ্বিতীয় বন্ধনীর ভেতরে লিখতে হয়। দ্বিতীয় বন্ধনীকে ইংরেজিতে বলে curly braces।

```
{laptop, cellphone, glass, notebook, pen}
```

আবার আমরা যেমন বলি, সপ্তম শ্রেণীর পাঠ্যবইয়ের একটি সেট। সেটের উপাদানগুলো যেকোনো ক্রমে লিখা যায়। যেমন, সপ্তম শ্রেণীর পাঠ্যবইয়ের সেটে আমরা বাংলা, ইংরেজি, গণিত, সমাজবিজ্ঞান … এভাবে লিখতে পারি, কিংবা, ইংরেজি, বাংলা, সমাজবিজ্ঞান, গণিত … এভাবেও লেখা যায়, তাতে কোনো সমস্যা নেই, সেট একই থাকছে। তবে সেটের কোনো উপাদান একাধিকবার লেখা যায় না। আমার টেবিলে যদি দুটি কিংবা আরো বেশি কলম থাকতো, তাহলেও আমি সেটের মধ্যে pen একবারই লিখতাম।

দুটো সেটের সবগুলো উপাদান মিলে নতুন সেট তৈরি করা যায়। এটিকে বলে ইউনিয়ন (union) করা, মানে, ইউনিয়ন এখানে একটি অপারেশন। যেমন : A = {1, 2, 3, 4, 5} ও B = {2, 4, 6, 8, 10} দুটি সেট হলে A ইউনিয়ন B হবে : {1, 2, 3, 4, 5, 6, 8, 10}। আবার দুটো সেটের সাধারন উপাদানগুলো অর্থাৎ যেসব উপাদান দুটো সেটেই আছে, তাদেরকে নিয়ে একটি নতুন সেট তৈরি করা যায়। যে অপারেশনের মাধ্যমে এটি করা হয়, তাকে বলে ইন্টারসেকশন (intersection)। A ও B-এর ইন্টারসেকশন করলে আমরা পাবো : {2, 4}।

পাইথনে সেট নিয়ে কাজ করার জন্য সেট নামেই একটি ডেটা স্ট্রাকচার আছে। যেমন আমরা একটি ফাঁকা সেট তৈরি করতে পারি এভাবে : A = set()। নিচে উদাহরণ দেওয়া হলো :

```
>>> A = set()
>>> A
set()
>>> type(A)
<class 'set'>
```

আবার আমাদের কাছে যদি ইতিমধ্যেই কিছু উপাদান থাকে, সেগুলো থেকেও সেট তৈরি করা যায়। যেমন :

```
>>> items = {"pen", "laptop", "cellphone"}
>>> items
{'cellphone', 'pen', 'laptop'}
>>> type(items)
<class 'set'>
>>>
```

আবার আমাদের কাছে যদি কোনো লিস্ট বা টাপল থাকে, সেই লিস্ট বা টাপল থেকেও সেট তৈরি করা যায়।

```
>>> li = [1, 2, 3, 4]
>>> tpl = (1, 2, 3)
>>> A = set(li)
>>> A
{1, 2, 3, 4}
>>> B = set(tpl)
>>> B
{1, 2, 3}
>>> type(A)
<class 'set'>
>>> type(B)
<class 'set'>
>>>
```

স্ট্রিং থেকেও সেট তৈরি করা যায়, সে ক্ষেত্রে স্ট্রিংয়ের প্রতিটি অক্ষর সেটের একটি উপাদান হবে। আর সেটের ক্ষেত্রে কিন্তু একটি জিনিস মনে রাখতে হবে, সেটের উপাদান আগে-পরে থাকতে পারে, কোনো ক্রম (order) অনুসরণ করতে হয় না।

```
>>> A = set("Bangladesh")
>>> A
{'e', 'g', 'h', 'a', 's', 'l', 'B', 'd', 'n'}
>>> type(A)
<class 'set'>
>>> B = set("Sri Lanka")
>>> B
{'i', 'k', 'L', ' ', 'r', 'a', 'S', 'n'}
>>> type(B)
<class 'set'>
>>>
```

কোনো জিনিস একটি সেটের সদস্য কি না, সেটি আমরা বের করতে পারি এভাবে :

```
>>> A = {1, 2, 3}
>>> 1 in A
True
>>> 2 in A
```

```
True
>>> 5 in A
False
```

এখন, আমার যদি দুটি সেট থাকে, আমি সেই দুটি সেটের ওপর বিভিন্ন অপারেশন চালাতে পারি। আমরা যদি চাই যে, A ও B দুটি সেটের সাধারণ উপাদানগুলো নিয়ে একটি সেট C তৈরি করবো, তাহলে আমরা লিখতে পারি : C = A & B (মানে ইন্টারসেকশন করা)। আর আমরা যদি ইউনিয়ন করতে চাই, অর্থাৎ, A কিংবা B যেকোনো একটি সেটের সদস্য (সেসব সদস্য আবার দুটো সেটের মধ্যেও থাকতে পারে), তাহলে লিখবো C = A | B। আবার আমরা যদি চাই, A কিংবা B, যেকোনো একটি সেটে আছে কিন্তু একই সঙ্গে দুটি সেটে নেই, সেসব সদস্য পেতে, তখন লিখবো C = A ^ B। আর যদি এমন হয় যে, আমরা চাই যেসব সদস্য A সেটে আছে কিন্তু B সেটে নেই, তাদের পাওয়া যাবে এভাবে : C = A – B। নিচের উদাহরণ দেখলে আমরা বুঝতে পারবো :

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {2, 4, 6, 8}
>>> C = A & B
>>> C
{2, 4}
>>> C = A | B
>>> C
{1, 2, 3, 4, 5, 6, 8}
>>> C = A ^ B
>>> C
{1, 3, 5, 6, 8}
>>> C = A - B
>>> C
{1, 3, 5}
>>> C = B - A
>>> C
{8, 6}
```

ডিকশনারি (Dictionary)

আমাদের যদি এখন একটি কাজ করতে হয় যে আমরা একটি ক্লাসের সকল শিক্ষার্থীর বাংলা পরীক্ষায় প্রাপ্ত নাম্বার রাখবো, তাহলে আমরা সেটি কিভাবে করতে পারি? আমরা তাহলে একটি লিস্ট ব্যবহার করতে পারি, যেখানে লিস্টের প্রথম উপাদান হবে ক্লাসের যেই শিক্ষার্থীর রোল নম্বর 1, তার পরীক্ষার নম্বর, দ্বিতীয় উপাদান হবে, যেই শিক্ষার্থীর রোল নম্বর 2, তার প্রাপ্ত নম্বর, এরকম। যেমন আমরা এভাবে লিস্টে রাখতে পারি :

```
marks = [78, 72, 68, 80, 63, 75]
```

তাহলে যার রোল নম্বর 3, তার বাংলা পরীক্ষায় প্রাপ্ত নম্বর হবে marks[2]। নিচের প্রোগ্রামটি একটি ফাইলে সেভ করে রান করতে হবে :

```
marks = [77, 76, 65, 78, 62, 64, 60, 77, 75, 79]
```

```
roll = input("Please enter your roll number: ")

print("Your marks is", marks[int(roll)-1])
```

রান করলে, আউটপুট আসবে এরকম :

```
$ python marks.py
Please enter your roll number: 4
Your marks is 78
```

আবার, আমাদের যদি এরকম করতে হত যে, বাংলা ও ইংরেজি, দুটি বিষয়ের পরীক্ষায় প্রাপ্ত নম্বর রাখতে হবে। তখন কিভাবে করবো? তখন আমরা প্রতি শিক্ষার্থীর জন্য একটি লিস্ট রাখতে পারি, যেখানে প্রথম উপাদান হচ্ছে বাংলা ও দ্বিতীয় উপাদান হচ্ছে ইংরেজি পরীক্ষায় প্রাপ্ত নম্বর। আর সকল শিক্ষার্থীর নম্বরের লিস্টগুলোকে রোল নম্বরের ক্রম (ক্রম মানে সিরিয়াল) অনুযায়ী সাজাতে পারি। যেমন :

```
>>> marks = [[74, 73], [70, 75], [68, 72], [73, 73]]
>>> marks[0]
[74, 73]
>>> marks[1]
[70, 75]
>>> marks[1][0]
70
>>> marks[1][1]
75
```

এভাবে কোড লিখার সমস্যা হচ্ছে, অনেককিছু আমাদেরকে মনে রাখতে হবে। যেমন, আমি যদি জানতে চাই যে রাফি ইংরেজি পরীক্ষায় কত নম্বর পেয়েছে, তখন আমাদের জানতে হবে যে রাফির রোল নম্বর কত, আর লিস্টে ইংরেজি কত নম্বর অবস্থানে আছে, সেটিও জানা থাকতে হবে। তো এই সমস্যা থেকে উদ্ধার পাওয়ার জন্য পাইথনে একটি চমৎকার ডেটা স্ট্রাকচার আছে, যার নাম ডিকশনারি।

ডিকশনারিতে ডেটা থাকে জোড়ায় জোড়ায়। জোড়ার প্রথম জিনিসটিকে বলে কি (key) আর দ্বিতীয় জিনিসটিকে বলে ভ্যালু (value)। কোনো ডেটা পেতে হলে, আমাদের কি (key) জানতে হয়, তাহলে আমরা ভ্যালু থেকে ডেটা উদ্ধার করতে পারি। যেমন, আমি যদি বাংলা পরীক্ষার নম্বরগুলো ডিকশনারিতে রাখি, তাহলে এভাবে রাখতে পারি :

```
>>> marks = {1: 77, 2: 76, 5: 62, 4: 78, 3: 65}
>>> type(marks)
<class 'dict'>
>>> marks[3]
65
>>> print("Marks of roll number 4 is", marks[4])
Marks of roll number 4 is 78
>>>
```

আমরা দেখতে পেলাম, কিভাবে ডিকশনারি ব্যবহার করতে হয়। ডিকশনারির টাইপ হচ্ছে dict সেটিও দেখলাম। আবার কোনো কি (key) জানলে ভ্যালু কিভাবে পেতে হয়, সেটিও দেখলাম। আরেকটি বিষয় খেয়াল করতে হবে যে, ডিকশনারিতে আমাদের কিন্তু কোনো ক্রম অনুসরণ করতে হয় না। আরেকটি মজার ব্যাপার হচ্ছে, রোল নম্বর যদি 1 থেকে শুরু না হয়ে 1001 থেকে শুরু হতো, তাহলেও কোনো সমস্যা ছিল না।

```
>>> marks = {1005: 75, 1003: 72, 1002: 70, 1001: 75, 1004: 77}
>>> marks[1003]
72
```

আবার এমন যদি হয়, রোল নম্বরটি আসলে সংখ্যা নয়, তাহলেও সমস্যা নেই। যেমন –

```
>>> marks = {"DH2001": 75, "DH1777": 72, "KH2050": 70}
>>> marks["DH2001"]
75
```

আমরা চাইলে একটি খালি ডিকশনারি (যেটিতে কোনো ডেটা নেই) তৈরি করতে পারি। কারণ, ডিকশনারি তৈরি করার পরেও এতে নতুন কি-ভ্যালু যোগ করা যায়।

```
>>> dt = {}
>>> dt
{}
>>> print(dt)
{}
>>> type(dt)
<class 'dict'>
>>> dt[1] = "one"
>>> print(dt)
{1: 'one'}
>>> dt[2] = "two"
>>> print(dt)
{1: 'one', 2: 'two'}
```

কোনো কি (key) যদি ডিকশনারিতে না থাকে, তাহলে সেটি একসেস করার সময় পাইথন এরর দেবে।

```
>>> dt = {"a": "A", "b": "B", "c": "C"}
>>> dt["a"]
'A'
>>> dt["b"]
'B'
>>> dt["c"]
'C'
>>> dt["d"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'd'
>>> dt["d"] = "D"
```

```
>>> dt["d"]
'D'
```

ডিকশনারিতে মিউটেবল নয়, এরকম যেকোনো কিছু কি (key) হিসেবে ব্যবহার করা যায়। যেমন: সংখ্যা, স্ট্রিং, টাপল। কিন্তু লিস্ট, সেট ব্যবহার করা যাবে না, কারণ লিস্ট তো মিউটেবল, মানে পরিবর্তন করা যায়। সেটও তেমনি, মিউটেবল হওয়ায় ডিকশনারির কি হিসেবে ব্যবহার করা যায় না। নিচের উদাহরণে আমরা এগুলো দেখব।

```
>>> dt = {"a": "A", "b": "B", "c": "C"}
>>> dt[(1, 2, 3)] = "tuple"
>>> dt
{(1, 2, 3): 'tuple', 'b': 'B', 'd': 'D', 'c': 'C', 'a': 'A'}
>>>
>>> li = [1, 2, 3]
>>> dt[li] = "list"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>>
>>> s = {1, 2, 3, 4}
>>> s
{1, 2, 3, 4}
>>> type(s)
<class 'set'>
>>> dt[s] = "set"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>>
```

এখন পর্যন্ত ডিকশনারির যেসমস্ত উদাহরণ দেওয়া হয়েছে সেগুলো নিজে নিজে চর্চা করতে হবে। এছাড়া মনে কোনো প্রশ্ন জাগলে কোড লিখে সেই প্রশ্নের উত্তর খুঁজে বের করার চেষ্টা করতে হবে।

অনুশীলনী :

- আমি যদি পরীক্ষার ফল রাখার জন্য একটি ডিকশনারি ব্যবহার করি আর সেখানে কি (key) হিসেবে শিক্ষার্থীর নাম ব্যবহার করি, তাহলে কী রকম সমস্যা সৃষ্টি হতে পারে?আমরা কিন্তু এখন ডিকশনারি ব্যবহার করে বিভিন্ন শিক্ষার্থীর বাংলা ও ইংরেজি পরীক্ষার নম্বর এভাবে রাখতে পারি :

```
>>> marks = {"DH1001": {"Bangla": 74, "English": 73}, "DH1002": {"Bangla":
70, "English": 75}}
>>> print(marks["DH1001"])
{'Bangla': 74, 'English': 73}
>>> print(marks["DH1001"]["English"])
73
>>> marks["DH1003"] = {"Bangla": 68, "English": 72}
>>> print(marks)
{'DH1003': {'Bangla': 68, 'English': 72}, 'DH1002': {'Bangla': 70, 'English':
75}, 'DH1001': {'Bangla': 74, 'English': 73}}
>>> print(marks["DH1003"])
```

```
{'Bangla': 68, 'English': 72}
>>> print(marks["DH1003"]["Bangla"])
68
```

এখন কিন্তু কোড লিখতে ও বুঝতে অনেক সহজ হয়ে গেল। আমাদের রোল নম্বর জানা থাকলে শিক্ষার্থীর সব বিষয়ের নম্বর দেখাতে পারি। আবার বিষয়ের নাম জানা থাকলে সেই বিষয়ে প্রাপ্ত নম্বরও দেখানো যায় সহজেই।

প্রোগ্রামিং হচ্ছে চর্চার বিষয়, আর তাই আমরা এখন ডিকশনারি ব্যবহারের আরো একটি উদাহরণ দেখবো। আমি যদি বাংলাদেশের সবগুলো বিভাগ সম্পর্কে কিছু তথ্য, যেমন, বিভাগের কয়টি জেলা, উপজেলা ও ইউনিয়ন রয়েছে, এগুলো নিয়ে কাজ করতে চাই, তাহলে পাইথনে ডিকশনারি হচ্ছে আদর্শ ডেটা স্ট্রাকচার। উইকিপিডিয়া থেকে আমি নিচের তথ্য সংগ্রহ করলাম –

Headquarter	Subdivision		
	District	Upazila	Union Council
Barisal	6	39	333
Chittagong	11	97	336
Dhaka	13	93	1,833
Khulna	10	59	270
Mymensingh	4	34	350
Rajshahi	8	70	558
Rangpur	8	58	536
Sylhet	4	38	334

এখন আমি ওপরের তথ্যগুলো একটি ডিকশনারিতে রাখতে পারি :

```
>>> bd_division_info = {}
>>> bd_division_info["Barishal"] = {"district": 6, "upazila": 39, "union": 
333}
>>> bd_division_info["Chittagong"] = {"district": 11, "upazila": 97, "union": 
336}
>>> bd_division_info["Dhaka"] = {"district": 13, "upazila": 93, "union": 
1833}
>>> bd_division_info["Khulna"] = {"district": 10, "upazila": 59, "union": 
270}
>>> bd_division_info["Mymensingh"] = {"district": 4, "upazila": 34, "union": 
350}
>>> bd_division_info["Rajshahi"] = {"district": 8, "upazila": 70, "union": 
558}
>>> bd_division_info["Rangpur"] = {"district": 8, "upazila": 58, "union": 
536}
>>> bd_division_info["Sylhet"] = {"district": 4, "upazila": 38, "union": 334}
>>> 
>>> print(bd_division_info)
{'Sylhet': {'upazila': 38, 'union': 334, 'district': 4}, 'Barishal': 
{'upazila': 39, 'union': 333, 'district': 6}, 'Rangpur': {'upazila': 58, 
'union': 536, 'district': 8}, 'Rajshahi': {'upazila': 70, 'union': 558, 
'district': 8}, 'Chittagong': {'upazila': 97, 'union': 336, 'district': 11}, 
'Mymensingh': {'upazila': 34, 'union': 350, 'district': 4}, 'Dhaka': 
{'upazila': 93, 'union': 1833, 'district': 13}, 'Khulna': {'upazila': 59, 
'union': 270, 'district': 10}}
>>>
```

এখন আমি যদি চাই সবগুলো বিভাগের নাম প্রিন্ট করবো, তাহলে কী করবো? বিভাগের নামগুলো ডিকশনারির কি (key) হিসেবে ব্যবহার করা হয়েছে। সেগুলো পাওয়ার জন্য পাইথনে একটি মেথড রয়েছে .keys()।

```
>>> divisions = bd_division_info.keys()
>>> print(divisions)
dict_keys(['Sylhet', 'Barishal', 'Rangpur', 'Rajshahi', 'Chittagong', 
'Mymensingh', 'Dhaka', 'Khulna'])
```

আমরা চাইলে এই কিগুলোর উপর লুপ চালিয়ে একে একে প্রিন্ট করতে পারি:

```
>>> for division in divisions:
...     print("Division", division)
... 
Division Sylhet
Division Barishal
Division Rangpur
Division Rajshahi
Division Chittagong
Division Mymensingh
Division Dhaka
Division Khulna
```

এখন যদি আমরা চাই, প্রতিটি বিভাগে কয়টি উপজেলা রয়েছে, সেই তথ্য প্রিন্ট করবো, তাহলে আমরা লুপ চালিয়ে কাজটি করতে পারি :

```
>>> for division in divisions:
...     print(division, "upazila", bd_division_info[division]["upazila"])
...
Sylhet upazila 38
Barishal upazila 39
Rangpur upazila 58
Rajshahi upazila 70
Chittagong upazila 97
Mymensingh upazila 34
Dhaka upazila 93
Khulna upazila 59
>>>
```

আমরা যদি সরাসরি ডিকশনারির উপর লুপ চালাই, তাহলে কেবল কি-গুলো পাবো:

```
>>> for item in bd_district_info:
...     print(item)
...
Sylhet
Barishal
Rangpur
Rajshahi
Chittagong
Mymensingh
Dhaka
Khulna
```

আমরা যদি কি ও ভ্যালু দুটোই পেতে চাই, তাহলে আমাদের হাতে দুটি পদ্ধতি রয়েছে। নিচের কোড দেখলেই বুঝতে পারা যাবে :

প্রথম পদ্ধতি:

```
>>> for key in bd_district_info:
...     print(key)
...     print(bd_district_info[key])
...
Sylhet
{'upazila': 38, 'union': 334, 'district': 4}
Barishal
{'upazila': 39, 'union': 333, 'district': 6}
Rangpur
{'upazila': 58, 'union': 536, 'district': 8}
Rajshahi
{'upazila': 70, 'union': 558, 'district': 8}
Chittagong
{'upazila': 97, 'union': 336, 'district': 11}
Mymensingh
{'upazila': 34, 'union': 350, 'district': 4}
```

```
Dhaka
{'upazila': 93, 'union': 1833, 'district': 13}
Khulna
{'upazila': 59, 'union': 270, 'district': 10}
```

দ্বিতীয় পদ্ধতি:

```
>>> for key, value in bd_district_info.items():
...     print(key)
...     print(value)
...
Sylhet
{'upazila': 38, 'union': 334, 'district': 4}
Barishal
{'upazila': 39, 'union': 333, 'district': 6}
Rangpur
{'upazila': 58, 'union': 536, 'district': 8}
Rajshahi
{'upazila': 70, 'union': 558, 'district': 8}
Chittagong
{'upazila': 97, 'union': 336, 'district': 11}
Mymensingh
{'upazila': 34, 'union': 350, 'district': 4}
Dhaka
{'upazila': 93, 'union': 1833, 'district': 13}
Khulna
{'upazila': 59, 'union': 270, 'district': 10}
```

অনুশীলনী

- bd_district_info ডিকশনারি ব্যবহার করে, বাংলাদেশে মোট কয়টি জেলা, উপজেলা ও ইউনিয়ন রয়েছে, সেই তথ্য বের করে প্রিন্ট করতে হবে।

আমরা এতক্ষণ পাইথনে যেসব ডেটা স্ট্রাকচার তৈরি করে দেওয়া আছে, সেগুলোর ব্যবহার দেখলাম। কখন কোন ডেটা স্ট্রাকচার ব্যবহার করতে হবে, সেটি নিজে থেকে বুঝে নিতে হবে। আর যখন আমাদের অভিজ্ঞতা আরো বাড়বে, তখন আমরা ডেটা স্ট্রাকচারগুলো আরো বেশি ভালোভাবে ব্যবহার করতে পারবো। অভিজ্ঞতা বাড়ানোর উপায় হচ্ছে অনুশীলন করা ও নতুন নতুন প্রোগ্রাম করা। এজন্য সময় ও শ্রম দেওয়া লাগে। নতুনদের এগুলো শিখতে অনেক সময় লাগে, তাই কোনো কিছু না পারলে চিন্তিত কিংবা হতাশ হওয়ার কিছু নেই, চেষ্টা চালিয়ে যেতে হবে।

মডিউল ও প্যাকেজ

Published by subeen on April 12, 2018

প্রোগ্রাম লেখার সময় আমাদের অনেক রকম কাজ করতে হতে পারে। কী কী কাজ করতে হবে, তা নির্ভর করে আমরা আসলে প্রোগ্রাম দিয়ে কী করতে চাচ্ছি তার উপরে। যেমন, আমরা গাণিতিক সমস্যার সমাধানের জন্য প্রোগ্রাম লিখতে পারি, অপারেটিং সিস্টেমের সঙ্গে কাজ করার জন্য প্রোগ্রাম লিখতে পারি, ইন্টারনেটে যুক্ত হয়ে ডেটা আদান-প্রদানের জন্য প্রোগ্রাম লিখতে পারি, ছবি-ভিডিও ইত্যাদি নিয়ে কাজ করতে পারি, কোনো গেমস তৈরি করতে পারি। এরকম কত কী যে করা যায় প্রোগ্রামিং করে, তার হিসেব নেই। এখন প্রোগ্রামারদের কাজ সহজ করার জন্য অনেক মডিউল তৈরি করে দেওয়া আছে। আবার একাধিক মডিউল একসঙ্গে করে প্যাকেজও তৈরি করে দেওয়া আছে। মডিউলগুলোতে থাকে বিভিন্ন ডেটা, ফাংশন ও ক্লাস (class)। ক্লাস কী জিনিস, সেটি আমরা এই বইতে পরে শিখবো।

আমরা যে বিভিন্ন প্রোগ্রামে print(), input(), type(), sum() ইত্যাদি ফাংশন ব্যবহার করেছি, সেগুলো কিন্তু আমরা নিজেরা তৈরি করি নি, ব্যবহার করেছি কেবল। এগুলোকে বিল্টইন (builtin) ফাংশন বলে। বিল্টইন শব্দের অর্থ হচ্ছে, যেগুলো কোনোকিছুর সঙ্গে দেওয়া থাকে। যেমন, টেলিভিশন অন-অফ করার জন্য বিল্টইন সুইচ দেওয়া থাকে। ওই ফাংশনগুলো তৈরি করা না থাকলে আমাদের নিজেদের সেগুলো তৈরি করা লাগত এবং প্রতিবার আমাদের বেশ খানিকটা সময় ব্যায় হতো। ফাংশনগুলো ইতিমধ্যে তৈরি করা আছে builtins নামক মডিউলে। পাইথন ইন্টারপ্রেটার চালু করলে এই মডিউল আপনাআপনি চলে আসে, নইলে আমাদেরকে আলাদাভাবে ইমপোর্ট (import) করতে হতো। কারো যদি জানতে ইচ্ছে করে যে, এই মডিউলে কী কী আছে, তাহলে সেটি জানার জন্য পাইথন ইন্টারপ্রেটার চালু করে নিচের দুই লাইন লিখতে হবে :

```
>>> import builtins
>>> dir(builtins)
```

dir() নিজেও একটি বিল্টইন ফাংশন, যার ভেতরে কোনো মডিউলের নাম আর্গুমেন্ট হিসেবে পাঠালে ওই মডিউলে কী কী গ্লোবাল ভ্যারিয়েবল, ফাংশন, ক্লাস ইত্যাদি আছে এগুলো সে একটি লিস্টে রিটার্ন করে।

স্ট্যান্ডার্ড লাইব্রেরি (Standard Library)

বিল্টইন ফাংশন ছাড়াও কিন্তু আমরা বিভিন্ন মডিউল ইমপোর্ট করে ব্যবহার করেছি। যেমন, math মডিউল ইমপোর্ট করেছি মৌলিক সংখ্যা বের করার প্রোগ্রাম লেখার সময়। এজন্য আমাদের লিখতে হয়েছে, import math। এই মডিউলটি হচ্ছে পাইথনের স্ট্যান্ডার্ড লাইব্রেরির একটি মডিউল। এখানে বিভিন্ন গাণিতিক ফাংশন তৈরি করে দেওয়া আছে, সেই সঙ্গে কিছু গাণিতিক ধ্রুবক (constant)-ও সংজ্ঞায়িত করে দেওয়া আছে। কিছু উদাহরণ দেখি :

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.pow(2, 3)
8.0
>>> math.pow(3, 2)
9.0
>>> math.sqrt(25)
5.0
>>> math.floor(5.2)
5
>>> math.ceil(5.2)
6
```

math মডিউলের কী কী ফাংশন আছে এবং সেগুলোর কাজ কী, তা পুরোপুরিভাবে জানতে হলে ডকুমেন্টেশন দেখতে হবে। পাইথনের ওয়েবসাইটে ডকুমেন্টেশন দেওয়া আছে (https://docs.python.org/3/library/math.html\) আবার চাইলে সেটি ডাউনলোডও করে ফেলা যায়। math ছাড়াও আমরা ইতিপূর্বে পাইথন দিয়ে প্রোগ্রামিং শেখা বইতে turtle, random, timeit ইত্যাদি মডিউল ব্যবহার করেছি। আরো কয়েকটি উদাহরণ দেখাই।

দিন-তারিখ নিয়ে কাজ-কর্ম করার জন্য আছে datetime মডিউল। এই মডিউলের মধ্যে আবার বিভিন্ন ক্লাস আছে। আমরা সেই ক্লাসের বিভিন্ন মেথড ব্যবহার করতে পারি। এখন আমরা যদি আজকের তারিখ জানতে চাই, তাহলে datetime মডিউলের date ক্লাসের today() মেথডকে কল করতে হবে।

```
>>> import datetime
>>> today = datetime.date.today()
>>> print(today)
2017-06-10
```

আবার আমরা যদি আজকের তারিখ ও বর্তমান সময় জানতে চাই, তখন datetime মডিউলের datetime ক্লাসের today() মেথড কল করতে হবে।

```
>>> import datetime
>>> today = datetime.datetime.today()
>>> print(today)
2017-06-10 15:51:28.951772
```

ওপরের দুটি উদাহরণে আমরা দেখলাম প্রথমে মডিউলের নাম, তারপরে একটি ডট, তারপরে ক্লাসের নাম, তারপরে একটি ডট ও সবশেষে ফাংশনের নাম লিখে আমরা ফাংশনটি কল করছি। আবার মডিউলটি যদি কোনো প্যাকেজের মধ্যে থাকত, তাহলে প্রথমে সেই প্যাকেজের নাম লিখে ডট দিতে হতো এবং তারপরে মডিউলের নাম, ক্লাসের নাম, ফাংশনের নাম ইত্যাদি লিখতে হতো। অর্থাৎ, package_name.module_name.function_name() আর যদি মডিউলে কোনো ক্লাসের মেথডকে কল করি, তাহলে package_name.module_name.class_name.function_name()।

আগের উদাহরণে আমরা চাইলে datetime মডিউল থেকে কেবল datetime ক্লাসটি ইমপোর্ট করতে পারতাম। তখন datetime.today() লিখলেই কাজ হয়ে যাবে।

```
>>> from datetime import datetime
>>> d = datetime.today()
>>> print(d)
2017-06-10 17:08:56.428959
```

আমরা যদি আমাদের প্রোগ্রাম থেকে কোনো ওয়েবসাইট, ওয়েব ব্রাউজারের মাধ্যমে দেখাতে চাই, তার জন্যও ফাংশন তৈরি করে দেওয়া আছে webbrowser মডিউলে।

```
>>> import webbrowser
>>> url = "http://subeen.com"
>>> webbrowser.open(url)
True
```

webbrowser মডিউলের open() ফাংশনটি ব্যবহার করে আমরা ওয়েবসাইট ওপেন করতে পারি। এখন webbrowser শব্দটি পুরোটা বার বার টাইপ করা অনেকের কাছে বিরক্তিকর লাগতে পারে। তার জন্য একটি শর্টকাটও আছে, যাকে বলে এলিয়াসিং (aliasing)। নিচের উদাহরণ দেখলেই বুঝতে পারবো :

```
>>> import webbrowser as wb
>>> url = "http://subeen.com"
>>> wb.open(url)
True
```

পাইথনের স্ট্যান্ডার্ড লাইব্রেরিতে কী কী মডিউল ও প্যাকেজ আছে, তা জানা যাবে এখানে : https://docs.python.org/3/library/index.html। অবসর সময়ে এগুলো নিয়ে পড়াশোনা করা যেতে পারে, তবে তাড়াহুড়ো নেই। আর সব লাইব্রেরির ব্যবহারও শেখার বা মুখস্থ করার দরকার নেই। সময়ের সঙ্গে প্রয়োজনমতো আমরা এমনিতেই শিখে নেবো।

আমাদের মাঝে-মধ্যে এমন কাজ করার দরকার পরবে, যেগুলোর জন্য কেউ একজন মডিউল বা প্যাকেজ তৈরি করে রেখেছে, কিন্তু সেগুলো পাইথনের স্ট্যান্ডার্ড লাইব্রেরি অন্তর্ভুক্ত নয়। এগুলো বলে থার্ড পার্টি প্যাকেজ বা মডিউল। এগুলো ব্যবহার করতে হলে কম্পিউটারে আলাদাভাবে ইনস্টল করতে হয়। আমাদের দরকার মতো আমরা সেগুলো ইনস্টল করে নিবো। এই বইতে পরবর্তি অংশে আমরা দেখবো, কিভাবে থার্ড পার্টি মডিউল ইনস্টল ও ব্যবহার করতে হয়।

নতুন মডিউল তৈরি করা

পাইথন দিয়ে প্রোগ্রামিং শেখা বইয়ের শেষ উদাহরণে আমরা দেখেছিলাম, কিভাবে ফিবোনাচ্চি সংখ্যা বের করতে হয়। n-তম ফিবোনাচ্চি সংখ্যা বের করার জন্য আমরা এখন একটি ফাংশন লিখে ফেলি।

```
def find_fib(n):
    if n <= 2:
        return 1
    fib_x, fib_next = 1, 1

    i = 3
    while i <= n:
        i += 1
        fib_x, fib_next = fib_next, fib_x + fib_next

    return fib_next

for x in range(1, 11):
    print(find_fib(x))
```

আমরা find_fib() নামে একটি ফাংশন তৈরি করলাম, যেটির প্যারামিটার হচ্ছে n এবং ফাংশনটি n-তম ফিবোনাচ্চি সংখ্যা রিটার্ন করে। তারপর একটি লুপ চালিয়ে আমরা ওই ফাংশন কল করে প্রথম দশটি ফিবোনাচ্চি সংখ্যা প্রিন্ট করলাম। আউটপুট আসবে নিচের মতো – প্রথম দশটি ফিবোনাচ্চি সংখ্যা :

```
$ python fibo.py
1
1
2
3
5
8
13
21
34
55
```

অনুশীলনী :

- ওপরের প্রোগ্রামে return fib_next না লিখে return fib_x লিখলে কী সমস্যা হতো? এটি নিয়ে একটু চিন্তাভাবনা করতে হবে।
- একটি ফাংশন লিখতে হবে, যার কাজ হচ্ছে প্যারামিটার হিসেবে n নেওয়া এবং প্রথম n-সংখ্যক ফিবোনাচ্চি সংখ্যার লিস্ট রিটার্ন করা। ফাংশনটির নাম যদি হয় list_fib(), তাহলে print(list_fib(10)) লিখলে আউটপুট আসবে :

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

list_fib() ফাংশনটিও আমরা fibo.py ফাইলে লিখবো। অধিকাংশ পাঠকেরই ফাংশনটি লিখতে ঘণ্টাখানেক চেষ্টা করতে হতে পারে, তাতে হতাশ হওয়ার কিছু নেই। প্রোগ্রামিং শেখার সময় যে যত বেশি চেষ্টা করবে, চিন্তাভাবনা করবে, কোড লিখবে, তার প্রোগ্রামিং শেখা তত ভালো হবে। এক-দুই ঘণ্টা চেষ্টা করেও list_fib() ফাংশনটি লেখতে না পারলে সামনে আগাতে হবে, আমি ফাংশনটি একটু পরেই লিখে দিয়েছি।

তাহলে আমার fibo.py ফাইলটি দাঁড়াচ্ছে এরকম :

```
def find_fib(n):
    if n <= 2:
        return 1

    fib_x, fib_next = 1, 1

    i = 3
    while i <= n:
        i += 1
        fib_x, fib_next = fib_next, fib_x + fib_next

    return fib_next


def list_fib(n):
    fib_list = [1, 1]
    if n <= 2:
        return fib_list[:n]

    fib_x, fib_next = 1, 1

    i = 3
    while i <= n:
        i += 1
        fib_x, fib_next = fib_next, fib_x + fib_next
        fib_list.append(fib_next)

    return fib_list


for x in range(1, 11):
    print(find_fib(x))

print(list_fib(1))
print(list_fib(2))
print(list_fib(10))
```

এবারে প্রথম কাজ হচ্ছে প্রোগ্রামটি নিজ নিজ কম্পিউটারে টাইপ করে রান করে দেখা। আউটপুট আসবে এরকম :

```
$ python fibo.py
1
1
2
3
5
8
13
21
34
55
```

```
[1]
[1, 1]
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

আমি ফাইলের শেষে প্রিন্টগুলো করে নিশ্চিত হওয়ার চেষ্টা করেছি যে, ফাংশনদুটি ঠিকঠাক কাজ করে। এখন আমরা fibo.py ফাইলটি যে ডিরেক্টরি বা ফোল্ডারে আছে, সেখানে আরেকটি ফাইল তৈরি করবো, program.py আর সেখানে নিচের মতো কোড লিখবো :

```
import fibo

print("Hello, I am inside program.py!")

n = fibo.find_fib(15)

print("15th fibonacci number is,", n)
```

এই প্রোগ্রামে আমি আসলে কী করেছি? প্রথমে fibo.py ফাইলটি ইমপোর্ট করেছি। ইমপোর্ট করার সময় ফাইলের এক্সটেনশন .py দেওয়ার প্রয়োজন নেই, তাই আমরা লিখেছি import fibo। এটি করার উদ্দেশ্য হচ্ছে fibo.py ফাইলের ভেতরে যেই ফাংশন দুটি তৈরি করেছি, সেগুলো যেন আমি ব্যবহার করতে পারি। fibo.py ফাইলে দুটি ফাংশন আছে – find_fib() ও list_fib()। ফাংশনদুটিকে কল করতে হলে আমাদের লিখতে হবে, যথাক্রমে fibo.find_fib() ও fibo.list_fib()। প্রোগ্রামের তৃতীয় লাইনে লিখেছি n = fibo.find_fib(15), অর্থাৎ আমি 15-তম ফিবোনাচ্চি সংখ্যা বের করে, সেই সংখ্যাটি n-এর মধ্যে রাখছি। তারপরের লাইনে সেটি প্রিন্ট করছি। এখন আমরা প্রোগ্রামটি রান করে দেখি, আউটপুট কী আসে?

```
$ python program.py
1
1
2
3
5
8
13
21
34
55
[1]
[1, 1]
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
Hello, I am inside program.py!
15th fibonacci number is, 610
```

আমাদের প্রোগ্রামে দুটি লাইন প্রিন্ট করার কথা, সেগুলো ঠিকই প্রিন্ট হয়েছে। কিন্তু তার আগে আবার fibo.py প্রোগ্রামে যা যা প্রিন্ট করতে দিয়েছিলাম, সেগুলোও প্রিন্ট হয়েছে। কেন এমন হলো? একটু তদন্ত করে দেখা যাক। আমরা এখন আমাদের program.py ফাইলে, শেষ দুটি স্টেটমেন্ট বাদ দিয়ে দেবো। ফাইলটি হবে এরকম :

```
import fibo

print("Hello, I am inside program.py!")
```

এখন প্রোগ্রাম রান করলে আউটপুট আসবে এরকম :

```
$ python program.py
1
1
2
3
5
8
13
21
34
55
[1]
[1, 1]
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
Hello, I am inside program.py!
```

তাহলে আমরা বুঝতে পারছি, যখনই আমরা import fibo লিখছি, পাইথন করছে কী, fibo.py ফাইলে রান করছে, তাই ওই ফাইলের শেষে আমি যেই প্রিন্ট স্টেটমেন্টগুলো লিখেছিলাম, সেগুলোর আউটপুটও আমরা দেখতে পাচ্ছি।

এখানে fibo.py ফাইলটিকে বলা হয় মডিউল (module)। একটি পাইথন মডিউলে সাধারণত বিভিন্ন ভেরিয়েবল ও ফাংশন ডেফিনেশন থাকে। মডিউলের ভেতরে ক্লাসও থাকতে পারে (ক্লাস সম্পর্কে আমরা এই বইতে পরে জানবো)। মডিউলটি ইমপোর্ট করে তার ভেতরের জিনিসগুলো ব্যবহার করা যায়। একটু আগেই আমরা তা দেখেছি। কিন্তু একটি বিষয় আমাদের ভাবিয়ে তুলছে। আমরা চাই যে, fibo.py ফাইলটি যদি সরাসরি রান করি, তাহলে যেন তার ভেতরে ফাংশনগুলো কল করে যে প্রিন্টগুলো করেছিলাম, সেগুলো চলবে, কিন্তু অন্য ফাইল থেকে fibo-কে ইমপোর্ট করলে ওই স্টেটমেন্টগুলো চলবে না। এজন্য আমাদেরকে একটি গ্লোবাল (global) ভ্যারিয়েবলের সাহায্য নিতে হবে, যার নাম হচ্ছে, __name__। আর এই গ্লোবাল ভ্যারিয়েবলটি কিন্তু আমরা নিজেরা ডিক্লেয়ার করবো না, প্রোগ্রাম রান করার সময় পাইথন নিজে থেকেই এর মান ঠিক করে দেয়। এখন আমরা trial.py নামে একটি ফাইল তৈরি করে এক লাইন কোড লিখবো। fibo.py ও program.py ফাইলদুটি যে ডিরেক্টরিতে আছে, trial.py ফাইলটিও সেই ডিরেক্টরিতে রাখতে হবে।

```
print("My name is", __name__)
```

এখন প্রোগ্রামটি রান করি :

```
$ python trial.py
My name is __main__
```

মডিউলের নাম তো হওয়ার কথা ছিল trial, কিন্তু আউটপুটে দেখছি __main__। এর কারণ হচ্ছে আমি যখন trial.py ফাইলটি সরাসরি রান করছি, তখন পাইথন তাকে কোনো মডিউল হিসেবে বিবেচনা করছে না, তাই তার নাম দেখাচ্ছে __main__। এখন আমি program.py ফাইলে trial মডিউল ইমপোর্ট করবো।

```
import trial

print("Hello, I am inside program.py!")
print(trial.__name__)
```

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম :

```
$ python program.py
My name is trial
Hello, I am inside program.py!
trial
```

প্রথম লাইনে যখন import trial লিখেছি, তখন trial.py ফাইলের প্রিন্ট স্টেটমেন্ট রান করে আউটপুট দিয়েছে My name is trial। এখন কিন্তু সে তার মডিউলের নাম ঠিকঠাক প্রিন্ট করলো। কারণ অন্য কেউ তাকে ইমপোর্ট করেছে। তাহলে আমরা দেখতে পাচ্ছি যে বাইরে থেকে ইমপোর্ট করলে সঠিক নাম দেখায়, আর ফাইলটি সরাসরি চালালে নাম দেখায় __main__। এই ব্যাপারটি আমরা কাজে লাগিয়ে এমন ব্যবস্থা করতে পারি যেন trial.py সরাসরি রান করলে Hello from trial কথাটি প্রিন্ট হবে, কিন্তু বাইরে থেকে ইমপোর্ট করলে এটি প্রিন্ট হবে না। trial.py ফাইলের কোড হবে এরকম :

```
if __name__ == "__main__":
    print("Hello from trial")
```

প্রোগ্রামটি রান করি :

```
$ python trial.py
Hello from trial
```

এবারে program.py ফাইলটি রান করি :

```
$ python program.py
Hello, I am inside program.py!
trial
```

এখন কিন্তু আর trial.py ফাইলের print স্টেটমেন্টটি রান করছে না, কারণ if কন্ডিশনের ভেতরে __name__ -এর মান হচ্ছে trial, তাই শর্তটি মিথ্যা হয়ে যাচ্ছে।

অনুশীলনীঃ fibo.py ফাইলটি এমনভাবে পরিবর্তন করতে হবে, যেন ফাইলটি মডিউল আকারে ইমপোর্ট করলে ফাইলের শেষের প্রিন্টগুলো না দেখায়, কিন্তু fibo.py ফাইলটি সরাসরি রান করলে যেন প্রিন্টগুলো দেখায়।

আমরা এতক্ষণ সবগুলো ফাইল একই ডিরেক্টরিতে রেখে কাজ করেছি। তার কারণ কী? কোনো প্রোগ্রামে যখন কোনো মডিউল ইমপোর্ট করা হয়, তখন পাইথন কিভাবে জানবে যে, মডিউলের কোড কোন জায়গায় আছে? যখন কোনো মডিউল ইমপোর্ট করা হয় তখন পাইথন প্রথমে খুঁজে দেখে ওই নামে কোনো বিল্ট-ইন মডিউল আছে কি না। ওই নামে কোনো বিল্ট-ইন মডিউল না থাকলে তার সার্চ পাথের সবগুলো ডিরেক্টরি ও তাদের সাব-ডিরেক্টরির মধ্যে একে একে খুঁজতে থাকে যে, সেই মডিউলটি পাওয়া যায় কি না। সার্চ পাথ (search path) হচ্ছে মূলত অনেকগুলো ডিরেক্টরির একটি তালিকা বা লিস্ট। সার্চ পাথের তালিকার প্রথম এলিমেন্ট বা উপাদান হচ্ছে ”, যেটা বোঝায় বর্তমান ডিরেক্টরি (কারেন্ট ওয়ার্কিং ডিরেক্টরি – **c**urrent **w**orking **d**irectory বা cwd), যেখানে আমরা আমাদের প্রোগ্রামটি চালাচ্ছি। সার্চ পাথের সবগুলো ডিরেক্টরির লিস্ট দেখতে চাইলে আমাদেরকে নিচের কোড ব্যবহার করতে হবে।

```
>>> import sys
>>> sys.path
>>> # এখানে একটি ডিরেক্টরির লিস্ট আসবে
```

আমরা চাইলে আমাদের মডিউলটি সার্চ পাথের যেকোনো একটি ডিরেক্টরিতে রেখে দিতে পারি, তাহলে পরবর্তি সময়ে পাইথন সেটিকে সহজে খুঁজে পাবে। আবার যদি এমন হয় যে, আমি একটি মডিউল বা প্যাকেজ তৈরি করলাম, যা আরো অনেক প্রোগ্রামারের কাজে লাগতে পারে, তখন আমি সেই মডিউল বা প্যাকেজটি পাবলিশ করতে পারি। তবে এই বইতে আমি সেটি দেখাবো না। কারো প্রয়োজন হলে সে শিখে নেবে, আমি বিষয়টি জানিয়ে রাখলাম মাত্র।

নোট: একটি মডিউল যদি বেশ বড়সড় হয়ে যায়, তখন তাকে আবার বিভিন্ন আলাদা ফাইলে ভেঙ্গে একটি ডিরেক্টরির মধ্যে রেখে একটি প্যাকেজ তৈরী করা যায়। একটি মডিউলে যেমন, একাধিক ভ্যারিয়েবল, ফাংশন, ক্লাস ইত্যাদি থাকতে পারে; তেমনি, একটি প্যাকেজের মধ্যে একাধিক ফাইল বা মডিউল, সাব-ডিরেক্টরি (যাদেরকে সাব-প্যাকেজও বলা হয়) থাকতে পারে। প্যাকেজের মূল ডিরেক্টরির মধ্যে __init__.py নামে একটি ফাইল তৈরী করে তার মধ্যে বলে দিতে হয়, এই প্যাকেজের মধ্যে কী কী সাব-প্যাকেজ ও মডিউল রয়েছে।

অবজেক্ট ও ক্লাস

Published by subeen on April 11, 2018

এই অধ্যায়ে আমরা অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং সম্পর্কে জানবো। এর জন্য আমাদেরকে ফেরত যেতে হবে সেই টার্টলের কাছে। আমরা ইতিমধ্যে এই বইতে টার্টল মডিউলের বিভিন্ন ফাংশন কল করে বিভিন্ন রকমের কাজ করেছি। উদাহরণ হিসেবে আমি এখন টার্টল দিয়ে বৃত্ত তৈরি করার একটি প্রোগ্রাম লিখবো।

```
import turtle

turtle.circle(50)

turtle.done()
```

প্রোগ্রামটি রান করলে একটি বৃত্ত তৈরি হবে। তো এই ধরণের প্রোগ্রামিংকে বলা হয় স্ট্রাকচার্ড প্রোগ্রামিং (আমরা এই বইতে এবং তার আগের বইতে বেশিরভাগ সময়ই স্ট্রাকচার্ড প্রোগ্রামিং করেছি)। এখন আমরা এই প্রোগ্রামটিই করবো, কিন্তু সেজন্য turtle মডিউলের মধ্যে Turtle নামে একটি ক্লাস (class) আছে, সেটি ব্যবহার করে। আমরা Turtle ক্লাস ব্যবহার করে টার্টলের অবজেক্ট বা ইনস্ট্যান্স (instance) তৈরি করবো। এখন এই ক্লাস, অবজেক্ট, ইনস্ট্যান্স – এগুলো কী জিনিস?

প্রথমে বলে নিই, অবজেক্ট আর ইনস্ট্যান্স একই জিনিস। একেক সময় একেক নাম ব্যবহার করা হয়। আর যখনই অবজেক্ট (বা ইনস্ট্যান্স)-এর কথা বলা হয়, তার সঙ্গে আরেকটি জিনিস যুক্ত করে দিতে হয়, যে এই অবজেক্টটি কোন ক্লাসের অবজেক্ট? ক্লাস হচ্ছে মূল নকশা আর সেই ক্লাসের এক বা একাধিক অবজেক্ট তৈরি করা যায়, যারা ওই ক্লাসের সব বৈশিষ্ট্য ধারণ করবে। যেমন, একটি গাড়ির ডিজাইন হচ্ছে ক্লাস আর সেই ডিজাইন অনুসরণ করে যত গাড়ি তৈরি করা হয়, সেই গাড়িগুলো হচ্ছে ওই ক্লাসের অবজেক্ট। তেমনি turtle মডিউলের ভেতরে Turtle ক্লাসে বলা আছে যে, একটি টার্টলের কী কী বৈশিষ্ট্য থাকবে এবং সেটি কী কী কাজ করতে পারবে ও কিভাবে করতে পারবে। এই বৈশিষ্ট্যগুলোকে বলা হয় ডেটা অ্যাট্রিবিউট (data attribute) আর যেসব কাজ করতে পারবে, সেগুলোকে বলা হয় মেথড (method)। আসলে মেথডগুলো হচ্ছে ক্লাসের ভেতরে তৈরি করা ফাংশন, তবে ক্লাস ও অবজেক্টের বেলায় তাদেরকে আমরা মেথড বলি। একারণেই এই বইতে এবং আগের বইতে কোথাও কোথাও ফাংশন, কোথাও কোথাও মেথড শব্দ ব্যবহার করা হয়েছে।

অবজেক্ট তৈরি ও ব্যবহার

টার্টল ক্লাসের অবজেক্ট তৈরি করবো কীভাবে? পাইথন ইন্টারপ্রেটারে সেটি আমি দেখাচ্ছি।

```
>>> import turtle
>>> tom = turtle.Turtle()
```

প্রথমে আমরা turtle মডিউল ইমপোর্ট করছি। তারপর tom নামে Turtle ক্লাসের একটি অবজেক্ট তৈরি করছি। এরজন্য মডিউলের নাম (turtle), তারপরে একটি ডট (.), তারপরে ক্লাসের নাম (Turtle) এবং তারপরে প্রথম বন্ধনী ব্যবহার করেছি। পাইথনে ক্লাসের নামের প্রথম অক্ষর সাধারণত বড় হাতের (capital letter) হয়। এখন আমি tom নামের এই টার্টলকে দিয়ে বিভিন্ন কাজ করাতে পারবো। যেমন, আমি যদি 100 পিক্সেল ব্যাসার্ধের একটি বৃত্ত আঁকতে চাই, তাহলে লিখবো :

```
>>> tom.circle(100)
```

এখন আমি যদি দেখতে চাই যে, tom কী টাইপের (বা কোন ক্লাসের) অবজেক্ট, তাহলে আমাকে type() ফাংশন ব্যবহার করতে হবে।

```
>>> type(tom)
<class 'turtle.Turtle'>
```

আমরা দেখতে পাচ্ছি, tom হচ্ছে turtle মডিউলের অন্তর্গত Turtle ক্লাসের অবজেক্ট।

আমরা চাইলো আরো টার্টল অবজেক্ট তৈরি করতে পারি। আমি এখন আরো দুটি টার্টল ক্লাসের অবজেক্ট তৈরি করবো :

```
>>> nonte = turtle.Turtle()
>>> fonte = turtle.Turtle()
```

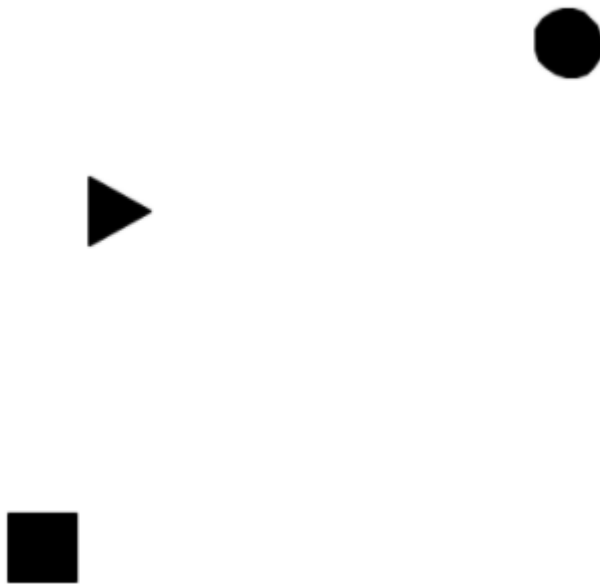
এখন আমি তাদের আকৃতি পরিবর্তন করে দেবো। তারপরে nonte-কে 30 ডিগ্রী বাম দিকে ঘুরে 100 পিক্সেল সামনে যেতে বলবো। আর Fonte-কে বলব 50 পিক্সেল পেছনে চলে আসতে।

```
>>> nonte.shape("circle")
>>> fonte.shape("square")
>>> nonte.left(30)
>>> nonte.forward(100)
>>> fonte.backward(50)
```

প্রতিটি লাইনের পরপর কিন্তু স্ক্রিন হালনাগাদ (আপডেট) হবে। সেটি সঙ্গে সঙ্গে দেখতে হবে। তাহলে বুঝতে পারা যাবে যে, কোন লাইন কী কাজ করছে। এখন আমি আরেকটি টার্টল অবজেক্ট তৈরি করবো এবং টার্টলগুলোকে আরো কিছু কাজ করতে বলবো।

```
>>> monte = turlte.Turtle()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'turlte' is not defined
>>> monte = turtle.Turtle()
>>> monte.setpos(-100, -100)
>>>
>>> monte.foward(30)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
AttributeError: 'Turtle' object has no attribute 'foward'
>>> monte.forward(30)
>>> monte.clear()
>>> monte.clear()
>>> fonte.clear()
>>> fonte.shape("triangle")
>>> monte.shape("square")
>>>
```

ওপরের স্টেটমেন্টগুলো চালালে শেষ পর্যন্ত স্ক্রিন নিচের মতো দেখাবে :

এখন ওপরে আমরা কিন্তু দুটি এরর দেখতে পাচ্ছি, একটি হচ্ছে NameError, আরেকটি হচ্ছে AttributeError। প্রথম এররটি বলছে, NameError: name 'turlte' is not defined, অর্থাৎ পাইথন ইন্টারপ্রেটার turlte নামে কোনো কিছু খুঁজে পায় নি (কারণ, আমি বানান ভুল করেছি)। আর দ্বিতীয় এররটি বলছে, AttributeError: 'Turtle' object has no attribute 'foward', অর্থাৎ Turtle ক্লাসে foward নামে কোনো কিছু নেই (কারণ, এখানেও আমি বানান ভুল করেছি)। তার মানে, আমরা কেবল সেসব জিনিসই ব্যবহার করতে পারবো, যেগুলো ক্লাসের মধ্যে বলে দেওয়া আছে।

এতক্ষণ আমরা শিখলাম, কিভাবে একটি ক্লাসের অবজেক্ট তৈরি করতে হয়, অবজেক্টগুলোকে কিভাবে ব্যবহার করতে হয়। আবার একটু ভুল করলে কী হয়, তাও দেখলাম। অবজেক্ট তৈরির একটি বড় সুবিধা হচ্ছে, ক্লাসের মধ্যে যেসব বৈশিষ্ট্যের কথা বলা আছে, আমরা যখন সেই ক্লাসের অবজেক্ট তৈরি করবো, তখন প্রতিটি অবজেক্টের জন্য সেই বৈশিষ্ট্যগুলো আলাদা হবে। তাই আমরা nonte, fonte, monte – এই তিনটি টার্টলকে তিন রকম আকৃতি দিতে পারি, তাদেরকে আলাদা রংও দিতে পারি। তাদেরকে বিভিন্ন কমান্ড দিয়ে আলাদা জায়গায় পাঠাতে পারি।

একজনের বৈশিষ্ট্য বা আচরণ অন্যজনের ওপর কোনো প্রভাব ফেলে না। nonte-কে 100 পিক্সেল সামনে যেতে বললে কেবল সে-ই সেই কাজটি করবে, বাকী সবাই নিজেদের মতো থাকবে।

একটি অবজেক্টের কী কী অ্যাট্রিবিউট (ডেটা অ্যাট্রিবিউট ও মেথড) আছে, সেটা জানতে চাইলে dir() ফাংশন ব্যবহার করতে হবে। এসব অ্যাট্রিবিউটের নাম দেখে অবশ্য খুব বেশি কিছু বোঝা যাবে না। বরং ডকুমেন্টেশন পড়লেই বিস্তারিত জানা যাবে। আমরা আগের বইতে লিস্ট তৈরি করা শিখেছিলাম। যখন কোনো লিস্ট তৈরি করা হয়, সেই লিস্ট আসলে list ক্লাসের একটি অবজেক্ট।

```
>>> li = [1, 10, 2, 20]
>>> type(li)
<class 'list'>
```

ওপরে দেখতে পাচ্ছি যে, li হচ্ছে list ক্লাসের অবজেক্ট। এখন এই li-এর কী কী অ্যাট্রিবিউট আছে, তাও আমরা জানতে পারি, এভাবে :

```
>>> dir(li)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

যেসব অ্যাট্রিবিউটগুলোর নাম আন্ডারস্কোর চিহ্ন (_) দিয়ে শুরু হয়েছে, সেগুলো আপাতত আমাদের বোঝার দরকার নেই। সেগুলো বাদ দিলে আমরা দেখতে পাচ্ছি, লিস্টের এই অ্যাট্রিবিউটগুলো : 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'। এগুলোর বেশ কিছু ব্যবহার আমরা আগের বইতে দেখেছি। এখন চাইলে নিজে নিজে পরীক্ষানিরীক্ষা করে দেখা যেতে পারে যে, কোন অ্যাট্রিবিউট কী কাজ করে। আর পাইথনের অফিশিয়াল ডকুমেন্টেশনেও লিস্টের বিভিন্ন মেথডগুলোর কাজের বর্ণনা লেখা আছে। যেহেতু li হচ্ছে list ক্লাসের একটি অবজেক্ট, তাই এর বিভিন্ন মেথড ব্যবহার করার সময় আমরা প্রথমে লিখবো অবজেক্টের নাম, তারপরে ডট, তারপরে মেথডের নাম। যেমন : li.sort()।

```
>>> li.sort()
>>> li
[1, 2, 10, 20]
```

নতুন ক্লাস তৈরি করা

আমরা যদি নতুন ধরণের অবজেক্ট তৈরি করতে চাই, তাহলে আমাদেরকে প্রথমে নতুন ক্লাস তৈরি করতে হবে। ধরা যাক, আমি একটি কার রেসিং গেমস বানাবো। সেই গেমসের একটি মূল বিষয় হচ্ছে কার (Car) বা গাড়ি। আমাকে অনেক কার অবজেক্ট তৈরি করতে হবে। তাই প্রথমে Car নামে একটি ক্লাস তৈরি করবো। এই ক্লাসে কী কী ডেটা অ্যাট্রিবিউট ও মেথড থাকতে পারে? এখন নিজে

নিজে একটি তালিকা তৈরি করার চেষ্টা করে দেখতে হবে। আমি একটি সহজ তালিকা দিচ্ছি। তবে, এটি কোনো পূর্ণাঙ্গ তালিকা নয়, অনেকের মাথায় অন্যকিছুও আসতে পারে।

ডেটা অ্যাট্রিবিউট:

- নাম (name)
- উৎপাদনকারী প্রতিষ্ঠান (manufacturer)
- রং (color)
- তৈরির সাল (year)
- ইঞ্জিনের ক্ষমতা (সিসি) (cc)

মেথড :

- ইঞ্জিন চালু করা (start)
- ব্রেক করা (brake)
- চালানো (drive)
- ডানে-বাঁয়ে ঘোরা (turn)
- গিয়ার পরিবর্তন (change gear)

এভাবে প্রথমে আমাদের ক্লাসের ডিজাইন করতে হবে। তারপর ক্লাস ডায়াগ্রাম (class diagram) তৈরি করতে হবে, তবে এই বইতে সেটি আমি দেখাবো না, তাই আপাতত এটি না শিখলেও চলবে। তারপরে আমরা পাইথনে ক্লাসটি তৈরি করবো। এখন প্রশ্ন হচ্ছে, আমরা কি করে বুঝবো যে আমাদেরকে একটি ক্লাস তৈরি করতে হবে? আমরা যখন সফটওয়্যার তৈরি করবো, এটি যেই সমস্যার সমাধানের জন্য তৈরি, সেটি নিয়ে চিন্তা করে বিশেষ্য পদগুলো (Noun) খুঁজে বের করতে হবে। দেখা যাবে এগুলোর বেশিরভাগের জন্যই আমাদের ক্লাস তৈরি করার দরকার হবে। যেমন, কার রেসিং গেমের ক্ষেত্রে, Car, Player, Track ইত্যাদি। আবার আমরা যদি স্কুল ম্যানেজমেন্ট করার জন্য একটি সফটওয়্যার তৈরি করি, তাহলে সেখানে Student, Teacher, Employee, ClassRoom, Subject, Result ইত্যাদি ক্লাস থাকতে পারে। সবগুলোই কিন্তু বিশেষ্য পদ, অর্থাৎ Noun।

পাইথনে ক্লাস তৈরি করতে হলে প্রথমে class লিখে একটি স্পেস দিয়ে ক্লাসের নাম লিখতে হয়, তারপরে সেই ক্লাসের ভেতরে বিভিন্ন স্টেটমেন্ট লেখা হয়। এখানেও ইনডেন্টেশন করতে হয়। ক্লাস ডিক্লেয়ার করার পরের লাইন থেকে যতটুকু কোড ক্লাসের ভেতরে থাকবে, তারা এক ট্যাব (tab) ডান দিক থেকে শুরু হবে।

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

ওপরে ClassName হচ্ছে ক্লাসের নাম। আমরা যেই নামের ক্লাস তৈরি করতে চাই, সেই নামটি সেখানে ব্যবহার করবো। আর স্টেটমেন্ট-এর জায়গায় বিভিন্ন ভ্যারিয়েবলে মান রাখা কিংবা মেথড তৈরি করা যাবে।

```
class Car:
    name = "Premio"
    color = "white"

    def start():
        print("Starting the engine")
```

ওপরে আমরা Car নামে একটি ক্লাস তৈরি করলাম। তারপরে সেখানে name ও color নামে দুটি ডেটা অ্যাট্রিবিউট রাখলাম, আর start নামে একটি মেথড তৈরি করলাম। এখন আমরা এই অ্যাট্রিবিউটগুলো ব্যবহার করতে পারবো। নিচের কোডে সেটি দেখানো হয়েছে :

```
class Car:
    name = "Premio"
    color = "white"

    def start():
        print("Starting the engine")


print("Name of the car:", Car.name)
print("Color:", Car.color)

Car.start()
```

প্রোগ্রামটি সেভ করে রান করলে নিচের মতো আউটপুট আসবে :

```
$ python car.py
Name of the car: Premio
Color: white
Starting the engine
```

[নোট: কার রেসিং গেম তৈরি করার সময় কিন্তু start() মেথড কেবল একটি লাইন প্রিন্ট করবে না, বরং গাড়ি চালু করার একটা শব্দ হবে, অ্যানিমেশন হবে। কিন্তু সেগুলো আমার এই বইতে আলোচনার বিষয় নয়। আমরা বরং অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং শেখার চেষ্টা করছি।]

এখন কেউ প্রশ্ন করতে পারে যে, সব গাড়ির রং তো সাদা হবে না, আর সব গাড়ির নামও প্রিমিও (Premio) হবে না। তাহলে কী করা যায়? আমরা ক্লাসের ভেতরে ওই ডেটা অ্যাট্রিবিউটগুলোয় ফাঁকা স্ট্রিং অ্যাসাইন করতে পারি। তারপরে সেগুলো পরিবর্তন করা যাবে।

```
class Car:
    name = ""
    color = ""
```

```
    def start():
        print("Starting the engine")


Car.name = "Axio"
Car.color = "Black"
print("Name of the car:", Car.name)
print("Color:", Car.color)

Car.start()
```

এখন প্রোগ্রামটি রান করে দেখব। আর বইটি কিন্তু কেবল পড়ে গেলে চলবে না, সবকিছু বইয়ের সঙ্গে সঙ্গে নিজেও কোড করে প্র্যাকটিস করতে হবে।

```
$ python car.py
Name of the car: Axio
Color: Black
Starting the engine
```

আমরা যদি ওপরের কোডে print(dir(Car)) লিখি, তাহলে Car ক্লাসের তিনটি অ্যাট্রিবিউট দেখতে পাবো : ‘color’, ‘name’, ‘start’। সেই সঙ্গে অবশ্য আরো অ্যাট্রিবিউট দেখাবে, যেগুলো _ দিয়ে শুরু, আপাতত সেগুলো নিয়ে চিন্তা করা দরকার নেই।

এখন পর্যন্ত আমরা সরাসরি Car ক্লাসের বিভিন্ন অ্যাট্রিবিউট ব্যবহার করেছি। এখন আমরা Car ক্লাসের অবজেক্ট তৈরি করবো। এই অবজেক্ট তৈরির কাজটিকে পাইথনের ভাষায় বলে ইনস্ট্যানশিয়েট (instantiate) করা। এজন্য অবজেক্টকে ইনস্ট্যান্সও বলা হয়ে থাকে।

```
class Car:
    name = ""
    color = ""

    def start():
        print("Starting the engine")

# creating a Car object
my_car = Car()
my_car.name = "Allion"
print(my_car.name)
my_car.start()
```

এখানে আমি প্রথমে Car ক্লাসের একটি অবজেক্ট তৈরি করলাম : my_car = Car()। তারপর, my_car – এর নাম দিয়ে দিলাম : my_car.name = “Allion” এবং সেটি প্রিন্ট করে দেখলাম। শেষ লাইনে my_car-কে চালু করার চেষ্টা করলাম, তার start() মেথড কল করে। কিন্তু এখানেই বিপত্তি। প্রোগ্রামটি যদি আমরা চালাই, তাহলে নিচের মতো আউটপুট দেখবো :

```
$ python car.py
Allion
Traceback (most recent call last):
```

```
  File "car.py", line 12, in <module>
    my_car.start()
TypeError: start() takes 0 positional arguments but 1 was given
```

নাম প্রিন্ট হলো ঠিকই, কিন্তু তারপরে এরর দিলো যে, start() takes 0 positional arguments but 1 was given। এর অর্থ হচ্ছে, start() মেথড যখন তৈরি করা হয়েছে, সেখানে সেটি আর্গুমেন্ট হিসেবে কোনো কিছু নেয় না (def start()), মানে 0টি আর্গুমেন্ট নেয়, কিন্তু আমরা 1টি আর্গুমেন্ট পাঠাচ্ছি (but 1 was given)! কিন্তু আমরা কোথায় একটি আর্গুমেন্ট পাঠালাম? আমরা তো লিখেছি my_car.start()। ব্র্যাকেটের ভেতরে কিছু লিখি নি। আমরা যখন কোনো অবজেক্টের মেথড কল করি (যেমন : my_car.start()), তখন সেই মেথডের ভেতরে অবজেক্টটি আপনাআপনি চলে যায়। কেন যায়, সেটি আমরা পরবর্তি সময়ে আলোচনা করবো। আমাদের এখন যেটি করতে হবে, সেটি হচ্ছে, def start()-এর বদলে লিখতে হবে def start(self)।

```
class Car:
    name = ""
    color = ""

    def start(self):
        print("Starting the engine")

my_car = Car()
my_car.name = "Allion"
print(my_car.name)
my_car.start()
```

এখন প্রোগ্রামটি রান করলে ঠিকঠাক চলবে। self-এর বদলে অন্যকিছু লিখলেও চলে, কিন্তু এই ক্ষেত্রে পাইথনের রীতি হচ্ছে self লেখা।

আমরা যখন একটি Car ক্লাসের অবজেক্ট তৈরি করছি, চাইলে তখনই আমরা সেই অবজেক্টের বিভিন্ন বৈশিষ্ট্য বলে দিতে পারি। প্রথমে আমরা নিচের কোডটি টাইপ করে রান করবো, তারপর সেটি আলোচনা করবো।

```
class Car:
    name = ""
    color = ""

    def __init__(self, name, color):
        self.name = name
        self.color = color

    def start(self):
        print("Starting the engine")

my_car = Car("Corolla", "White")
print(my_car.name)
print(my_car.color)
my_car.start()
```

প্রোগ্রামটি রান করলে ঠিকঠাক আউটপুট আসবে। এখানে আমি একটি নতুন মেথড ব্যবহার করেছি, __init__()। যখন কোনো ক্লাসের অবজেক্ট তৈরি করা হয়, তখন এই মেথডটি আপনাআপনি (অটোমেটিক) কল হয়। তাই যখন my_car = Car("Corolla", "White") স্টেটমেন্টটি চলে, তখন আসলে Car ক্লাসের __init__ মেথড কল হয়। সেই মেথডের প্রথম প্যারামিটার হচ্ছে self। এটি সবসময়ই দিতে হবে। তারপরে যেহেতু আমরা গাড়ির নাম ও রং সেট করে দিতে চাই, তাই name ও color নামে দুটি প্যারামিটার রাখছি। তারপরে ফাংশনের ভেতরে self.name-এ name অ্যাসাইন করছি, আর self.color-এ color অ্যাসাইন করছি। আমি যখন লিখছি self.name, সেটি বোঝাচ্ছে যে, যেই অবজেক্টটি আমি তৈরি করছি, তার name অ্যাট্রিবিউট। এখানে Car ক্লাসের ভেতরে যে name ডিক্লেয়ার করেছি (name = ""), self.name, আর init ফাংশনের name প্যারামিটার – তিনটি কিন্তু আলাদা। এখানে এসে ব্যাপারটা একটু এলোমেলো লাগতে পারে। তাই আমরা ওপরের কোডটি একটু পরিবর্তন করে লিখি।

```
class Car:
    def __init__(self, n, c):
        self.name = n
        self.color = c

    def start(self):
        print("Starting the engine")

my_car = Car("Corolla", "White")
print(my_car.name)
print(my_car.color)
my_car.start()
```

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম –

```
$ python car.py
Corolla
White
Starting the engine
```

এখানে যখন আমি my_car = Car("Corolla", "White") লিখছি, তখন __init__(self, n, c)-এর self-এ যাচ্ছে my_car অবজেক্টের রেফারেন্স, n-এ যাচ্ছে "Corolla", আর c-তে যাচ্ছে "White"। তারপর যখন self.name = n স্টেটমেন্ট এক্সিকিউট হচ্ছে, তখন my_car অবজেক্টের name নামে একটি অ্যাট্রিবিউট তৈরি হচ্ছে আর সেখানে n অ্যাসাইন হচ্ছে। একইভাবে color অ্যাট্রিবিউট তৈরি হয়ে সেখানে c-তে যা পাঠিয়েছিলাম, তা অ্যাসাইন হচ্ছে।

Car ক্লাসে যে দুটি ডেটা অ্যাট্রিবিউট তৈরি করেছিলাম (name ও color), সেগুলো কিন্তু আমি বাদ দিয়ে দিয়েছি। কারণ আমি অবজেক্ট তৈরি করার সময় init মেথডের ভেতরে self.name ও self.color যখন লিখছি, তখন সেই অবজেক্টের জন্য name ও color নামে অ্যাট্রিবিউট তৈরি হয়ে যাচ্ছে। একে বলে ইনস্ট্যান্স অ্যাট্রিবিউট, যা কেবল ওই ক্লাসের ইনস্ট্যান্সের (বা অবজেক্টের) থাকে। তবে এখন কিন্তু আর ক্লাসের নাম লিখে ডট দিয়ে name (ও color) একসেস করা যাবে না। আমরা যদি লিখি print(Car.name), তাহলে আমরা এরর পাবো : AttributeError: type object

‘Car’ has no attribute ‘name’। অর্থাৎ Car ক্লাসের name নামে কোনো অ্যাট্রিবিউট নেই। আমরা যখন এরকম লিখেছিলাম –

```
class Car:
    name = ""
    color = ""
```

তখন, এই name ও color–কে বলা হয় ক্লাস অ্যাট্রিবিউট। বেশিরভাগ সময়ই আমাদের এরকম অ্যাট্রিবিউট তৈরি করার দরকার পরবে না।

এখন ক্লাসের ভেতরে যেসব মেথড আছে, তারা যদি ওই ক্লাসের অবজেক্টের বিভিন্ন অ্যাট্রিবিউট একসেস করতে চায়, সেই কাজটি করতে পারবে, তবে অ্যাট্রিবিউটের নামের আগে অবশ্যই self লিখে ডট দিতে হবে, তাহলে পাইথন বুঝতে পারবে যে, অ্যাট্রিবিউটটি কোন অবজেক্টের সঙ্গে সংশ্লিষ্ট।

```
class Car:
    def __init__(self, n, c):
        self.name = n
        self.color = c

    def start(self):
        print("name: ", self.name)
        print("color: ", self.color)
        print("Starting the engine")

my_car = Car("Corolla", "White")
my_car.start()
```

ওপরের প্রোগ্রামে আমরা start() মেথডের ভেতরে অবজেক্টের নাম ও রং প্রিন্ট করেছি। এখন একটি মজার জিনিস দেখাই। self-এ যে অবজেক্ট পাঠানো হয়, তার প্রমাণ পাওয়া যাবে নিচের প্রোগ্রাম চালালে –

```
class Car:
    def __init__(self, n, c):
        self.name = n
        self.color = c

    def start(self):
        print("name: ", self.name)
        print("color: ", self.color)
        print("Starting the engine")

my_car = Car("Corolla", "White")

Car.start(my_car)
```

আমরা my_car অবজেক্ট তৈরি করলাম। তারপর, Car.start() মেথড কল করলাম কিন্তু আর্গুমেন্ট হিসেবে my_car অবজেক্ট ব্যবহার করলাম। এই প্রোগ্রামের আউটপুট কিন্তু হুবুহু আগের প্রোগ্রামের মতোই হবে। তবে বলে রাখা ভালো যে, সাধারণত এভাবে আমরা start() মেথড কল করবো না।

এখন আমরা চাইলে একাধিক কার অবজেক্ট তৈরি করতে পারি।

```
class Car:
    def __init__(self, n, c):
        self.name = n
        self.color = c

    def start(self):
        print("name: ", self.name)
        print("color: ", self.color)
        print("Starting the engine")

my_car1 = Car("Corolla", "White")
my_car1.start()
my_car2 = Car("Premio", "Black")
my_car2.start()
my_car3 = Car("Allion", "Blue")
my_car3.start()
```

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম :

```
$ python car.py
name:  Corolla
color:  White
Starting the engine
name:  Premio
color:  Black
Starting the engine
name:  Allion
color:  Blue
Starting the engine
```

প্রতিটি অবজেক্টের অ্যাট্রিবিউটগুলো কিন্তু আলাদা। যেমন, my_car1-এর color আর my_car2-এর color আলাদা। এটি হচ্ছে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের একটি বড় সুবিধা। আমরা যত খুশি অবজেক্ট তৈরি করতে পারি। প্রতিটি অবজেক্ট তার নিজস্ব অ্যাট্রিবিউট ধারণ করবে।

আরেকটি কথা বলা প্রয়োজন। আমরা যে __init__ মেথড ব্যবহার করছি, যা অবজেক্ট তৈরি হওয়ার সময় আপনাআপনি কল হয়, একে বলে কনস্ট্রাকটর (constructor)।

এখন ধরা যাক, Car ক্লাসের অবজেক্ট তৈরি করার পর আমার খেয়াল হলো, অবজেক্টের একটি ডেটা অ্যাট্রিবিউট দরকার, যেটি ক্লাসের মধ্যে নেই। আর আমি এখন ক্লাসটিও পরিবর্তন করতে

চাইছি না। কিন্তু তাতে কোনো সমস্যা নেই। আমি চাইলে যেকোনো সময় আমাদের অবজেক্টের সঙ্গে একটি ডেটা অ্যাট্রিবিউট জুড়ে দিতে পারবো।

```
class Car:
    def __init__(self, n, c):
        self.name = n
        self.color = c

    def start(self):
        print("Starting the engine")

car = Car("Corolla", "White")
car.year = 2017
print(car.name, car.color, car.year)
```

অনুশীলনীঃ আমরা শুরুতে Car ক্লাসের যে ডিজাইন করেছিলাম, আমি কিন্তু সেটি পুরোপুরি ইমপ্লিমেন্ট করি নি। এখন সেই ডিজাইন দেখে ক্লাসটি পুরোপুরি ইমপ্লিমেন্ট করতে হবে।